PCI Bus Power Management Interface Specification

Revision 1.1

December 18, 1998

Revision History

Revision	Issue Date	Comments
1.0	June 30, 1997	Original Issue.
1.1	December 18, 1998	Integrated the 3.3Vaux ECR.

DISCLAIMER

This PCI Bus Power Management Interface Specification is provided "as is" with no warranties whatsoever, including any warranty of merchantability, noninfringement, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. The PCI SIG disclaims all liability for infringement of proprietary rights, relating to use of information in this specification. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted herein.

Intel and Pentium are registered trademarks of Intel Corporation.

OnNow and Windows NT are trademarks and Microsoft, Windows, and Win32 are registered trademarks of Microsoft Corporation.

All other product names are trademarks, registered trademarks, or servicemarks of their respective owners.

Contents

Chapter 5 PCI Function Power Management States

Chapter 6 PCI Bridges and Power Management

Chapter 7 Power Management Events

Chapter 8 Software Support for PCI Power Management

Chapter 9 Other Considerations

Tables

Figures

Chapter 1
Introduction

Since its introduction in 1993, PCI has become a very popular bus. It is used in a wide variety of computer systems sold today ranging from laptops to large servers. Its bandwidth and efficient support for multiple masters has allowed it to sustain high performance applications; while at the same time, its low pin count and high integration factor has enabled very low cost solutions.

Power management in the current PC platform is performed by a combination of BIOS and System Management Mode (SMM) code utilizing hardware unique to each platform. While this strategy has successfully brought the PC platform into the mobile environment, it is beset with problems because of the fact that there is no standard way to truly determine when the system is busy and when it is actually idle. The operating system does have this information, so it makes sense to give it the responsibility for power management. The reason that this has not happened up to now is a lack of standards to provide the operating system with the required information that would allow it to control the hardware in a platform independent way. This specification addresses this need.

While the *PCI Local Bus Specification* is quite complete with a solid definition of protocols, electrical characteristics, and mechanical form factors, no provision was made in the original specification for supporting power management functionality. This specification addresses this requirement by defining four distinct power states for the PCI bus and four distinct power states for PCI functions as well as an interface for controlling these power states.

1.1 Goals of This Specification

The goal of this specification is to establish a standard set of PCI peripheral power management hardware interfaces and behavioral policies. Once established, this infrastructure enables an operating system to intelligently manage the power of PCI functions and buses.

Detailed Goals for PCI Power Management Interface:

- Enable multiple PCI function power levels
- Establish a standard for PCI function Wakeup Events
- Establish a standard for reporting power management capabilities
- Establish a standard mechanism for controlling a PCI function's power state
- Establish a standard mechanism for controlling a PCI bus's power state
- Minimal impact to the *PCI Local Bus Specification*
- Backwards compatible with *PCI Local Bus Specification, Revision 2.1* and *Revision 2.0* compliant designs
- Preserve the designer's ability to deliver differentiated products
- Provide a single architecture for all markets from mobile through server

Key Attributes of This Specification:

- Enhances the PCI bus's Plug and Play capabilities by comprehending power management
- Standardized power state definitions
- Standardized register interface in PCI Configuration Space
- Standardized Wake events

1.2 Target Audience

This document is intended to address the needs of several distinct audiences. Developers of PCI peripherals are the first audience. This document describes the hardware requirements of such devices to allow an operating system to manage the power of those devices. Developers of PCI host bridges, PCI to expansion bus bridges, PCI to PCI bus bridges, and PCI to CardBus bridges will also find information describing the requirements for such devices to implement operating system directed power management.

Software developers are also a targeted audience for this specification. Specifically, developers of operating systems and device drivers need to understand the power management interfaces presented by compliant devices to be able to manage them.

1.3 Overview/Scope

In order to implement a power managed system under the direction of the operating system, a large array of tightly coupled hardware and software ingredients needs to be defined and integrated. Figure 1 outlines, at a high level, this set of required architectural building blocks.

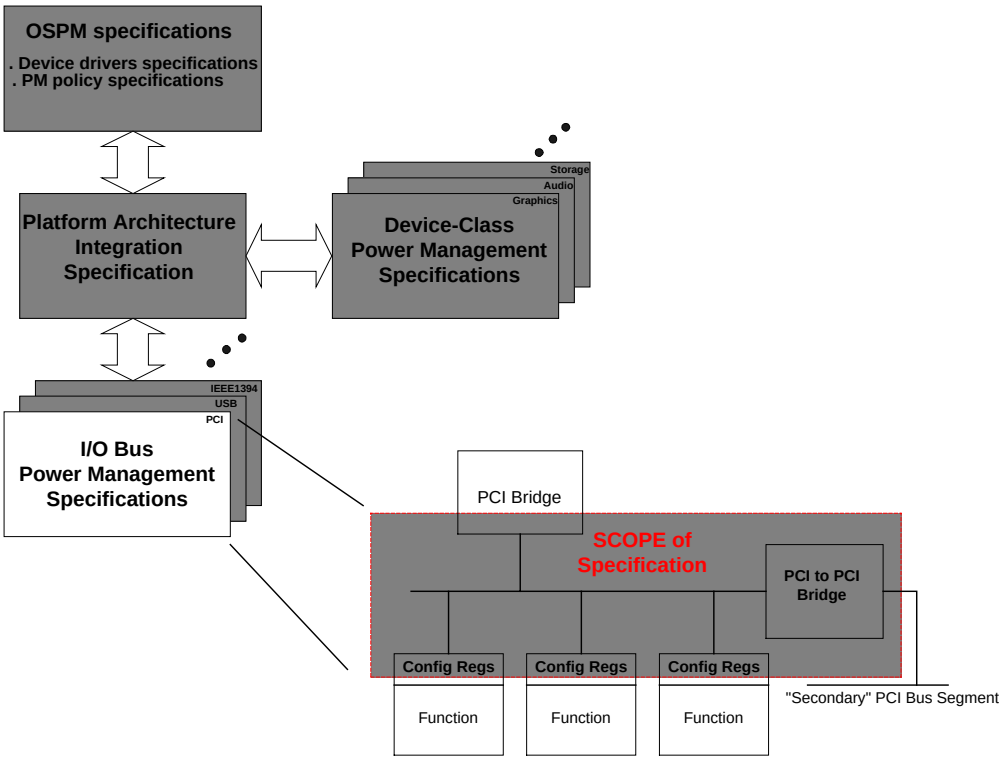


Figure 1: Operating System Directed Power Management System Architecture

The scope of this specification is sharply focused on establishing a standard set of interfaces that are required to power manage PCI bridges, buses, devices, and functions.

Any PCI based component can use the mechanisms described in this specification. This includes 32-bit and 64-bit PCI and 33-MHz and 66-MHz PCI devices.

Devices which can only be implemented on the motherboard are power managed in motherboard specific ways (such as ACPI) and, as such, fall outside the scope of this specification. Docking bridges, for example, fall into the motherboard devices category because the physical docking bridge component always resides logically on the motherboard and is never deployed as an add-in card. As such, docking bridges are not covered by this specification.

This document describes requirements for implementing power management for PCI functions which are capable of being used on an add-in card.

1.4 Glossary of Terms

3.3Vaux

3.3VDC auxiliary voltage supply. This auxiliary voltage supply is optionally provided by the PCI system to pin 14A of the PCI expansion connectors. **3.3Vaux** is used to power logic that needs to remain active when the rest of the system is unpowered (e.g., modem ring indicator detection logic). PCI functions that are enabled to draw current from this pin may consume no greater than 375 mA. Functions that have not been enabled to draw fully from this pin must draw no greater than 20 mA. Only PCI functions that support **PME#** generation from the $D3_{cold}$ state by design may utilize **3.3Vaux** power.

Bus Segment Reset

The hardware reset signal that is taken as actual physical input to a given component within a system. For example, the Bus Segment Reset signal for a PCI component is **RST#** as defined in the *PCI Local Bus Specification*.

Docking Bridge

Refers to a specialized motherboard device which provides bus expansion beyond the motherboard's base capabilities. This differs from a PCI to PCI bridge in that it can only be implemented as a motherboard device.

Legacy PCI Devices

A class of devices built before this specification which are *PCI Local Bus Specification, Revision 2.1* or *Revision 2.0* compliant. Legacy PCI devices are assumed to be in the ***D0*** power management state whenever power is applied to them.

Operating System

Throughout this specification, the terms operating system and system software refer to the combination of power management services, device drivers, user mode services, and/or kernel mode services.

Originating Device

From the perspective of the operating system (host CPU), the first bridge component encountered with a PCI bus downstream of it is defined as the Originating Device for that PCI bus segment. For example, consider a system supporting a host bridge and a PCI to PCI bridge whose primary bus is attached to the host bridge's PCI bus. In this configuration, the host bridge is defined as the Originating Device for the PCI bus that connects with the PCI to PCI bridge's primary bus, and the PCI to PCI bridge is defined as the Originating Device for its secondary PCI bus (see Figure 2).

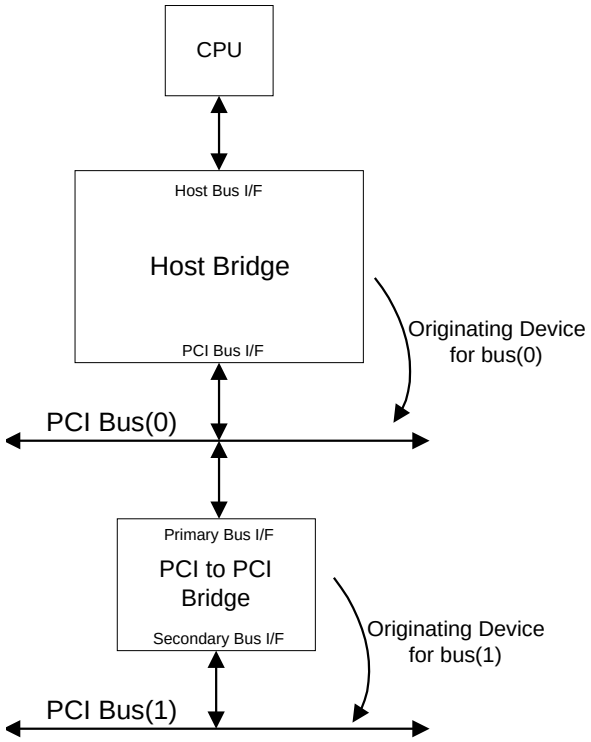


Figure 2: Example Originating Devices

PCI Bridge

PCI bridges couple two independent buses together. They are characterized by a primary bus interface and a secondary bus interface. There are several types of PCI bridges including host to PCI bridges, PCI to expansion bus bridges, PCI to PCI bridges, and PCI to CardBus bridges.

PCI Device

A physical device consisting of one load on the PCI bus and having only one **IDSEL** input. This single PCI device may contain up to eight PCI functions.

PCI Function	A set of functionality inside a PCI device represented by one 256-byte configuration space. Each PCI function within a device generally has a separate software driver.
PCI Function Context	The variable data held by the PCI function, usually volatile. Function context refers to small amounts of information held internal to the function. Function context is not limited only to the contents of the function's PCI registers, but rather refers also to the operational states of the function including state machine context, power state, execution stack (in some cases), etc.
PME Context	PME (Power Management Event) context is defined as the functional state information and logic required to generated power management events (PMEs), report PME status, and enable PMEs. PME Context specifically consists of the **PME_En** bit, **PME_Status** bit plus any device class specific event context (if any).
Power Management	Mechanisms in software and hardware to minimize system power consumption, manage system thermal limits, and maximize system battery life. Power management involves tradeoffs among system speed, noise, battery life, and AC power consumption.
Power Management Event (PME)	A Power Management Event (PME) is the process by which a PCI function can request a change of its power consumption state. Typically, a device uses a PME to request a change from a power savings state to the fully operational (and fully powered) state. However, a device could use a PME to request a change to a lower power state. A PME is requested via the assertion of the **PME#** signal. The power management policies of the system ultimately dictate what action is taken as a result of a PME.
Primary Bus	The primary bus of a PCI bridge refers to the bus that is topologically closest to the CPU that is running the operating system.
Restore Time	Restore time is defined as the time required to fully restore a PCI function to its fully operational state from a power saving mode of operation. It is measured as the total elapsed time between when the system software request for restoration occurs to when the function is fully configured and activated.

Secondary Bus

The secondary bus of a PCI bridge refers to the bus that is topologically farthest from the CPU that is running the operating system.

Sleeping State

A computer state where the computer consumes a small amount of power, user mode threads are *not* being executed, and the system appears to be off (from an end user's perspective, the display is off, etc.). Latency for returning to the Working state varies on the wakeup environment selected prior to entry of this state (for example, should the system answer phone calls, etc.). Work can be resumed without rebooting the operating system because large elements of system context are saved by the hardware and the rest by system software. It is not safe to disassemble the machine in this state.

Soft Off State

A computer state where the computer consumes a minimal amount of power. No user mode or system mode code is run. This state requires a large latency in order to return to the ***Working*** state. The system's context will not be preserved by the hardware. The system must be restarted to return to the ***Working*** state. It is not safe to disassemble the machine.

Wakeup Event

An event which can be enabled to wake the system from a ***Sleeping*** or ***Soft Off*** state to a ***Working*** state to allow some task to be performed.

Working State

A computer state where the system dispatches user mode (application) threads and they execute. In this state, devices (peripherals) are dynamically having their power state changed. The user will be able to select (through some user interface) various performance/power characteristics of the system to have the software optimize for performance or battery life. The system responds to external events in real time. It is not safe to disassemble the machine in this state.

1.5 Related Documents

PCI Local Bus Specification, Revision 2.1 and Revision 2.2, PCI Special Interest Group.

PCI to PCI Bridge Architecture Specification, Revision 1.0, April 5, 1994, PCI Special Interest Group.

PCI Mobile Design Guide, PCI Special Interest Group.

Advanced Configuration and Power Interface Specification, Revision 1.0, Intel Corporation, Microsoft Corporation, and Toshiba.

Instantly Available PC Power Management Design Guide, 1997, Intel Corporation.

Instantly Available PC Power Delivery Requirements and Recommendations, 1997, Intel Corporation.

OnNow Power Management and the Win32 Driver Model, 1996, Microsoft Corporation.

Device Class Power Management Reference Specifications, 1996, Microsoft Corporation.

PCI Hot-Plug Specification, PCI Special Interest Group.

1.6 Conventions Used in This Document

Several conventions are used in this document to help make it more readable. These are listed below:

- Power states are shown in bold italic text such as ***D0***.
- Register names are shown in italic text as in *PMCSR*.
- Names of bits or fields within registers are in bold text such as **PowerState**.
- Signal names are all capitalized, bold, and in a font such as **PME#**.
- All numbers are represented in decimal unless followed by a small letter.
  - Hexadecimal numbers are represented with a following “h” (e.g., DCh).
  - Binary numbers are represented with a following “b” (e.g., 10b).

Chapter 2
PCI Power Management Overview

2.1 PCI Power Management States

Power management states are defined as varying, distinct levels of power savings. Power management states are denoted by a state number. Throughout this document, power management states for both PCI buses and PCI functions will be defined, specified, and discussed. Power management states for PCI buses, by convention, are prefixed with a "***B***", and end in the power management state number ***(0-3).*** The higher the number, the more aggressive the intended power savings. Similarly, for PCI functions, the power management state is prefixed with a "***D***" and ends with the power management state number ***(0-3)***. Intended power savings increase with the power management state number.

2.1.1 PCI Function Power States

Up to four power states are defined for each PCI function in the system. These are ***D0-D3*** with ***D0*** being the maximum powered state, and ***D3*** being the minimum powered state. ***D1*** and ***D2*** power management states enable intermediate power savings states between the ***D0*** (on) and ***D3*** (off) power management states.

While the concept of these power states is universal for all functions in the system, the meaning, or intended functional behavior, when transitioned to a given power management state is dependent upon the type (or class) of the function.

2.1.2 Bus Power States

The power management state of a bus can be characterized by certain attributes of the bus at a given time, such as whether or not power is supplied, the speed of the clock, and what types of bus activities are allowed. These states are referred to as ***B0, B1, B2,*** and ***B3***.

This specification defines a mechanism that enables explicit control of a PCI bus's power and PCI clock as a function of the power management state of its Originating Device. The mechanism can be disabled (refer to Section 3.2.5 and Section 4.7.1).

2.1.3 Device-Class Specifications

This specification standardizes the power management hardware interface for the PCI bus and PCI components.

However, PCI functions belonging to different device classes may behave differently when operating in the same power management state. This is the notion of Device-Class specific power management policy. For example, the list of capabilities that an audio subsystem would support in a given power management state would most likely be different than the list of capabilities supported by a graphics controller operating in the same power management state.

While Device-Class Power Management specifications fall outside the scope of this specification, they are mentioned here to inform the reader of their important relationship to the interfaces defined in this document.

Each class of device must have its own class specific set of power management policies to define their intended behavior while in each of the power management states.

For a fully integrated power management system, these class-specific power management policies must also be standardized. Each major device type must have a Device-Class Power Management specification that all manufacturers can use as a guide to facilitate the seamless integration of their power managed products into a system.

Device-Class Power Management specifications will generally cover the following areas:

- **Device-Class power characteristics**
  Each class of PCI function should have a standard definition for each function power management state. This should include target power consumption levels, command response latencies, and state-change latencies. Implementation details for achieving these levels (such as whether an entire functional block is powered-off or the clock is stopped) might be important to a particular device class and, if so, should be specified. If any of these characteristics are in conflict with the requirements of the bus specifications, the function may not be able to implement that state on that particular bus.

- **Minimum Device-Class power capabilities**
  Each class of PCI function should have a standard set of power capabilities appropriate to the class. An example might be to require support either for all four power states or for some lesser number. Requirements might also be specified for accuracy and frequency of power status updates. Finally, there might be class-specific requirements for Wakeup capabilities. For example, all modems should be able to wake the PC from ***D1***, and so on.

- **Device-Class functional characteristics**
  Each class of PCI function should have a standard definition of the available subset of functioning capabilities and features in each power state. For example, the network adapter can receive but cannot transmit, or the sound card is fully functional except that the power amplifiers are off, and so on.

- **Device-Class Wakeup characteristics**
  Each class of PCI function should have a standard definition of its Wakeup policy including a recommended resume latency specification. This includes specifying the various class-specific events that can wake up the system and the power states from which the Wakeup can be signaled with implementation details and examples where appropriate.

2.1.4 Bus Support for PCI Function Power Management

Four base capabilities enable a robust power management solution. The capabilities are defined as:

- **Get Capabilities operation**
  This operation informs the operating system of the power management capabilities and features of a given function. It is performed as a part of the operating system's device enumeration[1] and uses the information it receives to help determine the power management policies to implement in the system. Information required from the function include which power states are implemented and the function's Wakeup capabilities.
- **Set Power State operation**
  This operation puts the function into a specific power management state and enables power management features based on the global system power management policy and the function's specific capabilities. This includes setting the function to wake the system from a sleeping state if certain events occur.
- **Get Power Status operation**
  This operation returns information about the current power state of the PCI function.
- **Wakeup operation**
  This is a mechanism for having a PCI function wake the system from a sleeping state on specified events.

While individual PCI functions must support only the first three capabilities with wakeup being optional, all basic power management operations must be supported by the bus architecture to ensure that PCI functions on the bus can be power managed.

For multi-function PCI components, there is a common portion of bus interface logic that physically binds each of the supported functions to the PCI bus. This common logic's power consumption is explicitly reported, if supported, using the *Data* register of Function 0. For further detail on the optional reporting of PCI function power consumption, refer to Section 3.2.6.

Control of the common logic is handled by the multi-function device in a software transparent fashion. For further power savings in a runtime environment, the enabling and disabling of some portion of this common PCI bus interface logic is the responsibility of the multi-function component hardware. This implicit power control, if implemented, may be based upon the power states of the functions behind it. As an example, one might consider a hardware administered logic control policy where the common logic cannot be internally powered down unless all of the functions hosted by the device have been placed in the $\boldsymbol{D3_{hot}}$ state first.

[1] The PCI function provides power management capabilities reporting through a standard register definition as specified in this document.

Chapter 3
PCI Power Management Interface

The four basic power management operations that have been defined are: Capabilities Reporting, Power Status Reporting, Setting Power State, and System Wakeup. Of these four capabilities, all are required of each function with the exception of wakeup event generation. This chapter describes the format of the registers in PCI Configuration Space that are used by these operations.

The Status and Capabilities Pointer (*Cap_ptr*) fields have been highlighted in Figure 3 to indicate where the PCI Power Management features appear in the standard Configuration Space Header.

Device ID		*Vendor ID*		00h
Status (with bit 4 set to 1)		*Command*		04h
Class Code			*Revision ID*	08h
BIST	*Header Type*	*Latency Timer*	*Cache Line Size*	0Ch
Base Address Registers				10h 14h 18h 1Ch 20h 24h
CardBus CIS Pointer				28h
Subsystem ID		*Subsystem Vendor ID*		2Ch
Expansion ROM Base Address				30h
Reserved			*Cap_Ptr*	34h
Reserved				38h
Max_Lat	*Min_Gnt*	*Interrupt Pin*	*Interrupt Line*	3Ch

Figure 3: Standard PCI Configuration Space Header Type 0

Software must have a standard method of determining if a specific function is designed in accordance with this specification. This is accomplished by using a bit in the PCI Status register to indicate the presence of the Capabilities List and a single byte in the standard PCI Configuration Space Header which acts as a pointer to a linked list of additional capabilities. Software can then traverse the list looking for the proper ID for PCI Power Management (*Cap_ID*=01h). If there is no Capabilities List (**Capabilities** bit in the *Status* register = 0) or if the list does not contain an item with the proper ID, the function does not support the PCI power management interface described in this document, and the operating system will assume that the function only supports the ***D0*** and ***D3$_{cold}$*** power management states. These legacy PCI functions may also require a device specific initialization sequence after any transition from ***D3$_{cold}$*** to ***D0*** (refer to Section 8.3).

3.1 Capabilities List Data Structure

The **Capabilities** bit in the *PCI Status* register (offset = 06h) indicates whether or not the subject function implements a linked list of extended capabilities. Specifically, if bit 4 is set, the *Cap_Ptr* register is implemented to give the offset in configuration space for the first item in the list.

Table 1: PCI Status Register

Bits	Default Value	Read/ Write	Description
15:05	--	--	Definition given in *PCI Local Bus Specification, Revision 2.1*
04	1b	Read Only	**Capabilities** - This bit indicates whether this function implements a list of extended capabilities such as PCI power management. When set, this bit indicates the presence of Capabilities. A value of 0 means that this function does not implement Capabilities.
03:00	0 h	Read Only	Reserved

The location of the *Capabilities Pointer* (*Cap_Ptr*) depends on the PCI header type. Refer to Section 3.1.1 for Header Type specific *Cap_Ptr* offsets.

Table 2: Capabilities Pointer - Cap_Ptr

Bits	Default Value	Read/ Write	Description
07:00	XXh	Read Only	**The *Cap_Ptr*** provides an offset into the function's PCI Configuration Space for the location of the first item in the Capabilities linked list. The Cap_Ptr offset is DWORD aligned so the two least significant bits are always "0"'s.

If a function does not implement any capabilities with IDs defined by the PCI SIG, the **Capabilities** bit in the *PCI Status* register (bit 4) should read as "0" and the *Cap_Ptr* register should be ignored. Values of 00h-3Fh are not valid values for the *Cap_Ptr*

because they point into the standard PCI header. A PCI function may choose any DWORD aligned offset ≥ 40h for a capability list item.

Figure 4 shows how the capabilities list is implemented. The *Cap_Ptr* gives the location of the first item in the list which is the PCI Power Management registers in this example (although the capabilities can be in any order). The first byte of each entry is required to be the ID of that capability. The PCI Power Management capability has an ID of 01h. The next byte is a pointer giving an absolute offset in the functions PCI Configuration Space to the next item in the list and must be DWORD aligned. If there are no more entries in the list, the *Next_Item_Ptr* must be set to 0 to indicate that the end of the linked list has been reached. Each capability can then have registers following the *Next_Item_Ptr*. The definition of these registers (including layout, size, and bit definitions) is specific to each capability. The PCI Power Management Register Block is defined in this specification.

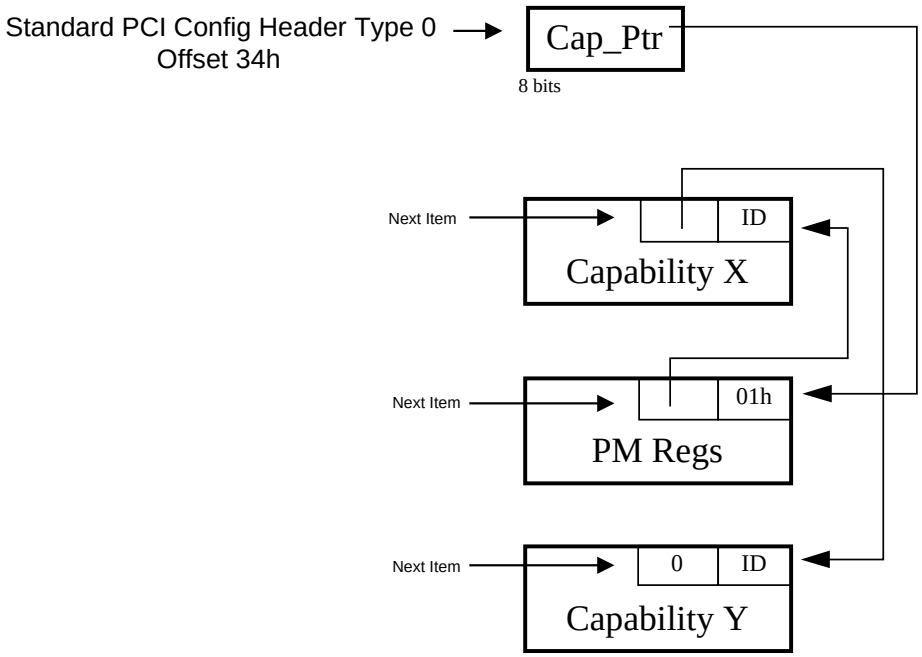


Figure 4: Capabilities Linked List

3.1.1 Capabilities List Cap_Ptr Location

There are currently three defined PCI Configuration Space Header types. Table 3 shows where to find the *Cap_Ptr* register for each of these Header Types.

Table 3: PCI Configuration Space Header Type/Cap_Ptr Mappings

Header Type	Associated PCI Function Type	Cap_Ptr (PCI Config Space Offset)
0	All other	34h
1	PCI to PCI bridge	34h
2	CardBus bridge	14h

Regardless of the implemented Header Type, the definition of the *Cap_Ptr* register and the **Capabilities** bit in the PCI *Status* register is common.

3.2 Power Management Register Block Definition

This section describes the PCI Power Management Interface registers.

Figure 5 illustrates the organization of the PCI Power Management Register Block. The first 16 bits (Capabilities ID [offset = 0] and Next Item Pointer [offset = 1]) are used for the linked list infrastructure.

The next 32 bits (*PMC* [offset = 2] and *PMCSR* registers [offset = 4]) are required for compliance with this specification. The next 8-bit register (bridge support *PMCSR* extensions [offset = 6]) is required only for bridge functions, and the remaining 8-bit *Data* register [offset = 8] is optional for any class of function. As with all PCI configuration registers, these registers may be accessed as bytes, 16-bit words, or 32-bit DWORDs.

Unless otherwise specified, all write operations to reserved registers must be treated as no-ops; that is, the access must be completed normally on the bus and the data discarded. Read accesses to reserved or unimplemented registers must be completed normally and a data value of 0 returned.

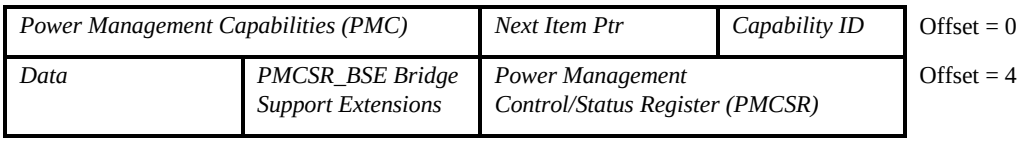


Figure 5: Power Management Register Block

The offset for each register is listed as an offset from the beginning of the linked list item which is determined either from the *Cap_Ptr* (if Power Management is the first item in the list) or the *Next_Item_Ptr* of the previous item in the list.

3.2.1 Capability Identifier - Cap_ID (Offset = 0)

The *Capability Identifier*, when read by system software as 01h indicates that the data structure currently being pointed to is the PCI Power Management data structure. Each function of a PCI device may have only one item in its capability list with *Cap_ID* set to 01h.

Table 4: Capability Identifier - Cap_ID

Bits	Default Value	Read/ Write	Description
07:00	01h	Read Only	**ID** - This field, when "01h" identifies the linked list item as being the PCI Power Management registers.

3.2.2 Next Item Pointer - Next_Item_Ptr (Offset = 1)

The *Next Item Pointer* register describes the location of the next item in the function's capability list. The value given is an offset into the function's PCI Configuration Space. If the function does not implement any other capabilities defined by the PCI SIG for inclusion in the capabilities list, or if power management is the last item in the list, then this register must be set to 00h.

Table 5: Next Item Pointer - Next_Item_Ptr

Bits	Default Value	Read/ Write	Description
07:00	00h	Read Only	**Next Item Pointer** - This field provides an offset into the function's PCI Configuration Space pointing to the location of next item in the function's capability list. If there are no additional items in the Capabilities List, this register is set to 00h.

3.2.3 PMC - Power Management Capabilities (Offset = 2)

The *Power Management Capabilities* register is a 16-bit read-only register which provides information on the capabilities of the function related to power management. The information in this register is generally static and known at design time.

Table 6: Power Management Capabilities - PMC

Bits	Default Value	Read/ Write	Description
15:11	Device Specific	Read Only	**PME_Support** - This 5-bit field indicates the power states in which the function may assert **PME#**. A value of 0b for any bit indicates that the function is not capable of asserting the **PME#** signal while in that power state. bit(11) XXXX1b - **PME#** can be asserted from ***D0*** bit(12) XXX1Xb - **PME#** can be asserted from ***D1*** bit(13) XX1XXb - **PME#** can be asserted from ***D2*** bit(14) X1XXXb - **PME#** can be asserted from $D3_{hot}$ bit(15) 1XXXXb - **PME#** can be asserted from $D3_{cold}$
10	Device Specific	Read Only	**D2_Support** - If this bit is a "1", this function supports the ***D2*** Power Management State. Functions that do not support ***D2*** must always return a value of "0" for this bit.
09	Device Specific	Read Only	**D1_Support** - If this bit is a "1", this function supports the ***D1*** Power Management State. Functions that do not support ***D1*** must always return a value of "0" for this bit.

Table 6: Power Management Capabilities – PMC (continued)

Bits	Default Value	Read/ Write	Description
08:06	Device Specific	Read Only	**Aux_Current -** This 3 bit field reports the **3.3Vaux** auxiliary current requirements for the PCI function. If the *Data Register* has been implemented by this function: • Reads of this field must return a value of "000b". • The Data Register takes precedence over this field for 3.3Vaux current requirement reporting. If **PME#** generation from ***D3$_{cold}$*** is not supported by the function (*PMC(15)*=0), this field must return a value of "000b" when read. For functions that support **PME#** from ***D3$_{cold}$***, and do not implement the *Data Register*, the following bit assignments apply : **Bit 3.3Vaux** **8 7 6 Max. Current Required** 1 1 1 375 mA 1 1 0 320 mA 1 0 1 270 mA 1 0 0 220 mA 0 1 1 160 mA 0 1 0 100 mA 0 0 1 55 mA 0 0 0 0 (self powered)
05	Device Specific	Read Only	**DSI** - The Device Specific Initialization bit indicates whether special initialization of this function is required (beyond the standard PCI configuration header) before the generic class device driver is able to use it. Note that this bit is not used by some operating systems. Microsoft Windows and Windows NT, for instance, do not use this bit to determine whether to use ***D3***. Instead, they use the driver's capabilities to determine this. A "1" indicates that the function requires a device specific initialization sequence following transition to the ***D0*** uninitialized state. Refer to Section 8.3.
04	0b	Read Only	Reserved

Table 6: Power Management Capabilities – PMC (continued)

Bits	Default Value	Read/ Write	Description
03	0b	Read Only	**PME Clock** - When this bit is a "1", it indicates that the function relies on the presence of the PCI clock for **PME#** operation. When this bit is a "0", it indicates that no PCI clock is required for the function to generate **PME#.** Functions that do not support **PME#** generation in any state must return "0" for this field.
02:00	010b	Read Only	**Version** - A value of 010b indicates that this function complies with Revision 1.1 of the *PCI Power Management Interface Specification.*

Note that bit 15 of the PMC (**PME#** can be asserted from $D3_{cold}$) represents a special case. Devices and functions which wish to set this bit will require some sort of auxiliary power source. These functions should ensure that the power source is available before setting this bit. This may be done at design time for motherboard devices receiving auxiliary power directly from the system, or for add-in cards, by predicating the state of this bit on the presence of voltage on the 3.3Vaux auxiliary power pin. Special consideration must be given to CardBus functions which receive auxiliary power on the same pins which provide Vcc. For a description of how auxiliary power is handled on CardBus devices, refer to the *PCI Style Power Management Interface Specification for CardBus Cards* white paper.

3.2.4 PMCSR - Power Management Control/Status (Offset = 4)

This 16-bit register is used to manage the PCI function's power management state as well as to enable/monitor PMEs.

The PME support bits, **PME_Status** and **PME_En**, are defined to be sticky bits for functions that can generate PMEs from $D3_{cold}$ in that their states are not affected by power on reset or transitions from $D3_{cold}$ to the ***D0*** Uninitialized state. Preservation of these bits is typically achieved by either powering them with an auxiliary power source or by using non-volatile storage cells for them. The only way to clear out these bits is to have system software write to them with the appropriate values.

As mentioned previously, the PME Context is defined as the logic responsible for identifying PMEs, the logic responsible for generating the **PME#** signal, and the bits within this register that provide the standard system interface for this functionality. PME Context also contains any device class specific status that must survive the transition to the ***D0*** Uninitialized state as well.

If a function supports **PME#** generation from $D3_{cold}$, its PME Context is not affected by either a PCI Bus Segment Reset (hardware component reset) or the internal "soft" re-initialization that occurs when restoring a function from $D3_{hot}$. This is because the function's PME functionality itself may have been responsible for the wake event which caused the transition back to ***D0***. Therefore, the PME Context must be preserved for the system software to process.

If **PME#** generation is not supported from $D3_{cold}$, then all PME Context is initialized with the assertion of a bus segment reset.

Because a PCI bus **RST#** assertion does not necessarily clear all functions' PME Context (functions that support **PME#** from $D3_{cold}$), the system software is required to explicitly initialize all PME Context, including the PME support bits, for all functions during initial operating system load. In terms of the *PMCSR*, this means that during the initial operating system load each function's **PME_En** bit must be written with a "0", and each function's **PME_Status** bit must be written with a "1" by system software as part of the process of initializing the system.

Table 7: Power Management Control/Status - PMCSR

Bits	Value at Reset	Read/ Write	Description
15	Sticky bit, indeterminate at time of initial operating system boot if function supports **PME#** from $D3_{cold}$. 0b, if the function does not support **PME#** from $D3_{cold}$.	Read/ Write-Clear	**PME_Status** - This bit is set when the function would normally assert the **PME**# signal independent of the state of the **PME_En** bit. Writing a "1" to this bit will clear it and cause the function to stop asserting a **PME**# (if enabled). Writing a "0" has no effect. This bit defaults to "0" if the function does not support **PME#** generation from $D3_{cold}$. If the function supports **PME#** from $D3_{cold}$, then this bit is sticky and must be explicitly cleared by the operating system each time the operating system is initially loaded.
14:13	Device Specific	Read Only	**Data_Scale** - This 2-bit read-only field indicates the scaling factor to be used when interpreting the value of the *Data* register. The value and meaning of this field will vary depending on which data value has been selected by the **Data_Select** field. This field is a required component of the *Data* register (offset 7) and must be implemented if the *Data* register is implemented. If the *Data* register has not been implemented, this field must return "00b" when the *PMCSR* is read. Refer to Section 3.2.6 for more details.

Table 7: Power Management Control/Status - PMCSR (continued)

Bits	Value at Reset	Read/ Write	Description
12:09	0000b	Read/ Write	**Data_Select** - This 4-bit field is used to select which data is to be reported through the *Data* register and **Data_Scale** field. This field is a required component of the *Data* register (offset 7) and must be implemented if the *Data* register is implemented. If the *Data* register is not implemented, this field may be hardwired to be read only always returning "0000b" when the *PMCSR* is read. Refer to Section 3.2.6 for more details.
08	Sticky bit, indeterminate at time of initial operating system boot if function supports **PME#** from ***D3***$_{cold}$. 0b**,** if the function does not support **PME#** from ***D3***$_{cold}$.	Read/ Write	**PME_En** - A "1" enables the function to assert **PME#.** When "0", **PME**# assertion is disabled. This bit defaults to "0" if the function does not support **PME#** generation from ***D3***$_{cold}$. If the function supports **PME#** from ***D3***$_{cold}$, then this bit is sticky and must be explicitly cleared by the operating system each time it is initially loaded. Functions that do not support **PME#** generation from any D-state (i.e., PMC(15:11) = "00000b"), may hardwire this bit to be read-only always returning a "0" when read by system software.
07:02	000000b	Read Only	Reserved
01:00	00b	Read/ Write	**PowerState** - This 2-bit field is used both to determine the current power state of a function and to set the function into a new power state. The definition of the field values is given below. 00b - ***D0*** 01b - ***D1*** 10b - ***D2*** 11b - ***D3***$_{hot}$ If software attempts to write an unsupported, optional state to this field, the write operation must complete normally on the bus; however, the data is discarded and no state change occurs.

3.2.5 PMCSR_BSE - PMCSR PCI to PCI Bridge Support Extensions (Offset = 6)

PMCSR_BSE supports PCI bridge specific functionality and is required for all PCI-to-PCI bridges.

Table 8: PMCSR Bridge Support Extensions - PMCSR_BSE

Bits	Value at Reset	Read/ Write	Description
07	External strap or internally hardwired	Read Only	**BPCC_En (Bus Power/Clock Control Enable)** - A "1" indicates that the bus power/clock control mechanism as defined in Section 4.7.1 is enabled. A "0" indicates that the bus power/clock control policies defined in Section 4.7.1 have been disabled. When the Bus Power/Clock Control mechanism is disabled, the bridge's *PMCSR* **PowerState** field cannot be used by the system software to control the power or clock of the bridge's secondary bus.
06	External strap or internally hardwired	Read Only	**B2_B3# (*B2/B3 support for D3$_{hot}$*)** - The state of this bit determines the action that is to occur as a direct result of programming the function to ***D3$_{hot}$***. A "1" indicates that when the bridge function is programmed to ***D3$_{hot}$***, its secondary bus's PCI clock will be stopped (***B2***). A "0" indicates that when the bridge function is programmed to ***D3$_{hot}$***, its secondary bus will have its power removed (***B3***). This bit is only meaningful if bit 7 (BPCC_En) is a "1". Refer to Section 4.7.1 for details.
05:00	000000b	Read Only	Reserved

3.2.6 Data (Offset = 7)

The *Data* register is an optional, 8-bit read-only register that provides a mechanism for the function to report state dependent operating data such as power consumed or heat dissipation. Typically the data returned through the *Data* register is a static copy (look up table, for example) of the function's worst case "DC characteristics" data sheet. This data, when made available to system software, could then be used to intelligently make decisions about power budgeting, cooling requirements, etc.

Any type of data could be reported through this register, but only power usage is defined by this version of the specification.

If the *Data* register is implemented, then the **Data_Select** and **Data_Scale** fields must also be implemented. If this register is not implemented, a value of 0 should always be returned by this register as well as for the **Data_Select** and **Data_Scale** fields.

Software may check for the presence of the *Data* register by writing different values into the **Data_Select** field, looking for non-zero return data in the *Data* register and/or **Data_Scale** field.

Any non-zero *Data* register / **Data_Select** read data indicates that the *Data* register complex has been implemented. Since **Data_Select** is a 4-bit field, an exhaustive presence detection scan requires 16 write/read operations to the **Data_Select** field and *Data* register / **Data_Scale** field respectively.

Table 9: Data Register

Bits	Default Value	Read/ Write	Description
07:00	00h	Read Only	**Data** - This register is used to report the state dependent data requested by the **Data_Select** field. The value of this register is scaled by the value reported by the **Data_Scale** field.

The *Data* register is used by writing the proper value to the **Data_Select** field in the *PMCSR* and then reading the **Data_Scale** field and the *Data* register. The binary value read from *Data* is then multiplied by the scaling factor indicated by **Data_Scale** to arrive at the value for the desired measurement. Table 10 shows which measurements are defined and how to interpret the values of each register.

Table 10: Power Consumption/Dissipation Reporting

Value in Data_Select	Data Reported	Data_Scale Interpretation	Units/Accuracy
0	***D0*** Power Consumed	0 = Unknown 1 = 0.1x 2 = 0.01x 3 = 0.001x	Watts
1	***D1*** Power Consumed		
2	***D2*** Power Consumed		
3	***D3*** Power Consumed		
4	***D0*** Power Dissipated		
5	***D1*** Power Dissipated		
6	***D2*** Power Dissipated		
7	***D3*** Power Dissipated		
8	Common logic power consumption (multi-function PCI devices, Function 0 only)		
9-15	Reserved (Function 0 of a multi-function device)	Reserved	TBD
8-15	Reserved (single function PCI devices and other functions (greater than Function 0) within a multi-function device)	Reserved	TBD

When using the *Data* register as a window into the data sheet for the PCI function, data returned must comply with measurements derived from the following test environment:

- Bus Frequency: 33 MHz/66 MHz (use 66-MHz characterization if the function is 66-MHz capable for the worst case data)
- V_{CC}: 5.25 VDC or 3.3 VDC (if 5 VDC is not supported)
- Temperature: 70 °C

The power measurements defined above have a dynamic range of 0 to 25.5 W with 0.1 W resolution, 0 to 2.55 W with 0.01 W resolution, or 0 to 255 mW with 1 mW resolution. Power should be reported as accurately as possible. For example, the data returned for each state supported must indicate the maximum power used by the function when in that particular state. The "Power Consumed" values defined above must include all power consumed from the PCI power planes through the PCI connector pins. If the PCI card provides power to external devices, that power must be included as well. It should not include any power derived from a battery or an external source. This information is useful for management of the power supply or battery.

The "Power Dissipated" values should provide the amount of heat which will be released into the interior of the computer chassis. This excludes any power delivered to external devices but must include any power derived from a battery or external power source and dissipated inside the computer chassis. This information is useful for fine grained thermal management.

If a function allows a wide range of implementation options, the values reported through this register may need to be loadable through a serial EPROM or strapping option at reset much like the *Subsystem Vendor ID* and *Subsystem ID* registers.

Multi-function devices implementing power reporting should report the power consumed by each function in each corresponding function's Configuration Space. In a multi-function device, the common logic power consumption is reported in function 0's Configuration Space through the *Data* register once the **Data_Select** field of the function 0's *PMCSR* has been programmed to 1000b. The sum of the values reported should then be accurate for the condition of all functions in the device being put in that state.

Multi-device cards implementing power reporting (i.e., multiple PCI devices behind a PCI bridge) should have the bridge report the power it uses by itself. Each function of each device on the card is responsible for reporting the power consumed by that function.

Chapter 4
PCI Bus Power States

This chapter describes the different power states of the PCI bus.

From a power management perspective, the PCI bus can be characterized at any point in time by one of four power management states. ***B0*** corresponds to the bus being fully useable (full power and clock frequency) and ***B3*** meaning that the power to the bus has been switched off. ***B1*** and ***B2*** represent intermediate power management states. The ***B1*** bus power management state is defined as a fully powered yet *enforced* idle[2] PCI bus with its clock free running. The ***B2*** state carries forward the characteristics of the ***B1*** state, but also has its clock stopped.

Table 11 shows a mapping of the four defined power states to key characteristics of the PCI bus.

Table 11: PCI Bus Power Management States

Bus States	Vcc	Clock	Bus Activity
B0 (Fully On)	On	Free running, PCI 2.1 compliant	Any PCI transaction, function interrupt, or PME event
B1	On	Free running, PCI 2.1 compliant	PME event
B2	On	Stopped, PCI 2.1 compliant[3] (in the low state)	PME event
B3 (Off)	Off	N/A (no power)	PME event

Each PCI bus in a system has an Originating Device which can support one or more power states. In most cases, this will be some kind of a bridge such as a host bus to PCI bus bridge or a PCI to PCI bridge.

[2] Enforced by the operating system which has previously programmed the functions residing on, and further downstream of, that particular bus segment to power management states that would preclude normal PCI transactions or functional interrupts from occurring.

[3] *PCI Local Bus Specification, Revision 2.1* compliant except that 66-MHz devices would normally require a full clock.

This specification defines a bus power (Vcc) and clock control mechanism (refer to Section 4.7.1) for a PCI bus assuming a single Vcc/Vss power plane pair[4], as well as a single logical clock that is shared by all devices on the bus. An implementation with slot specific or device specific control of Vcc and/or clocks is beyond the scope of this specification. System designers wanting to implement a device- or slot-specific Vcc power control structure should refer to the *PCI Hot-Plug Specification*.

4.1 PCI *B0* State - Fully On

All PCI buses support ***B0*** by default.

A PCI bus in ***B0*** is capable of running any legal PCI transaction.

Since ***B0*** is the only PCI bus power management state where data transactions can take place, system software must ensure that a PCI bus is in ***B0*** before attempting to access any PCI resources on that bus. If an access is attempted to a function residing downstream of a bus that is not in ***B0***, the transaction must be treated as if a Master Abort had occurred and error reporting handled in the same manner.

It is the system software's responsibility to ensure that, prior to attempting to program the bus to a power management state other than ***B0,*** all functions residing on that bus have previously been programmed to a state that would preclude any further bus activity initiated by them. For new PCI functions compliant with this specification, this means that all functions must have been previously programmed to a power management state that, for each of them, has precluded any bus activity on their parts. For legacy PCI functions, system software must rely on other means such as disabling the Bus Master Enable bit of the legacy function's *PCI Command* register to ensure that the function does not attempt to initiate any bus transactions.

The bus's Originating Device must always exit from ***B0*** gracefully by first allowing the bus to settle into the idle state.

4.2 PCI *B1* State

When a PCI bus is in ***B1***, Vcc is still applied to all devices on the bus; however, no bus transactions are allowed to take place on the bus. In effect, ***B1*** indicates a perpetual idle state on the PCI bus. All PCI bus signals are required to be held at valid logic states at all times, consistent with *PCI Local Bus Specification, Revision 2.1*. Note: Bus parking is allowed.[5]

The ***B1*** state, if comprehended by the bridge design (refer to Section 4.7), provides the bridge with information indicating that no functions residing on the bus will attempt to initiate any bus transactions, with the possible exception of a PME which does not go through the bridge. This information could then be used to intelligently apply more aggressive power savings in the bridge design.

---

[4] Single power plane pair that sources power to the Originating Device, its PCI bus, and all components logically residing on its PCI bus.

[5] In most cases, a function in a low power state (not ***D0***) will never have been the last mastering agent on the bus segment.

4.3 PCI *B2* State

When a PCI bus is in ***B2***, Vcc is still applied to all devices on the bus, but the clock is stopped and held in the low state. All PCI bus signals are required to be held at valid logic states at all times, consistent with *PCI Local Bus Specification, Revision 2.1*. Note: Bus parking is allowed.

The ***B2*** state, if comprehended by the bridge design (refer to Section 4.7), provides the bridge function with information indicating that no functions residing on the bus will attempt to initiate any bus transactions, with the possible exception of a PME. Nor do any of the PCI functions require a PCI clock. This information could then be used to intelligently apply more aggressive power savings in the bridge design. There is a minimum time requirement of 50 ms which must be provided by system software between when the bus is switched from ***B2*** to ***B0*** and when a function on the bus is accessed to allow time for the clock to start up and the bus to settle.

4.4 PCI *B3* State - Off

In ***B3,*** Vcc has been removed from all devices on the PCI bus segment. When Vcc is reapplied to the PCI bus, **RST#** must be asserted for that bus segment and the bus brought to an active, idle state in accordance with *PCI Local Bus Specification, Revision 2.1*.

B3 is exhibited by all *PCI Local Bus Specification, Revision 2.1* compliant systems when their power is removed. A programmable interface for placing a bus in ***B3***, or restoring it from ***B3,*** is optional.

4.5 PCI Bus Power State Transitions

The PCI bus power states can be changed as shown in Figure 6.

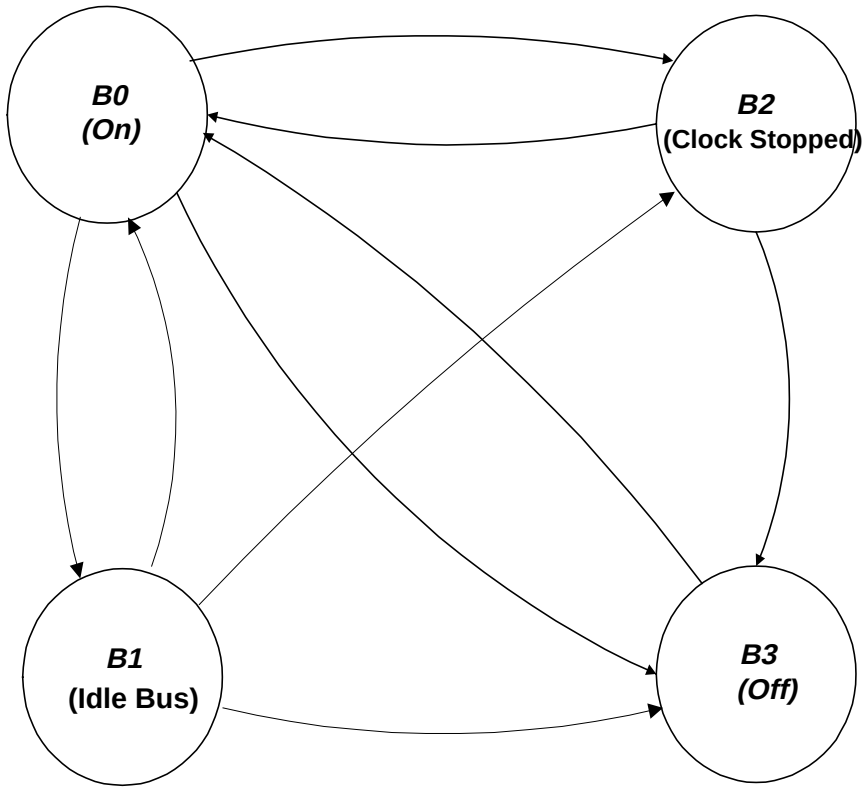


Figure 6: PCI Bus Power State Transitions

A system reset always returns the PCI bus to ***B0***.

Removing power always takes the bus to ***B3***. All other bus state changes are made by software which writes to the appropriate location in the bus's Originating Device. These programmed function power state transitions implicitly have impact on the next power state for a particular bus. All buses support ***B3*** by default if power is removed from the system. A bridge may optionally support ***B3*** when its power state is programmed to $D3_{hot}$.

4.6 PCI Clocking Considerations

PCI bridge functions must either directly drive and control the clock to their secondary bus or provide sideband information to an external clock source for that bus segment.

Unless otherwise specified, all PCI bus clocking is required to be *PCI Local Bus Specification, Revision 2.1* compliant.

4.6.1 Special Considerations for 66-MHz PCI Designs

The *PCI Local Bus Specification, Revision 2.1* adds several unique requirements on the clock for 66-MHz operation. The key 66-MHz device requirement from a power management point of view is that if the PCI clock is operating at a frequency over 33 MHz (i.e., 66 MHz), the speed of the PCI clock is not allowed to change except when **RST#** is active.

Designers who wish to implement 66-MHz PCI devices that fully comply with this specification's 33-MHz clocking requirements while operating at 66 MHz are encouraged to do so. However, 66-MHz capable PCI devices, while running at 66 MHz, that comply with the *PCI Local Bus Specification, Revision 2.1* 66-MHz clocking requirements are also compliant with this specification provided that all other requirements are met.

For functions that comply with the *PCI Local Bus Specification, Revision 2.1* 66-MHz clock specification, the power management capabilities reported to the system must indicate that its clock may not be stopped.

This must be done in the following way:

PMC bit(10), **D2_Support**, must always return a "0" when read from a 66-MHz capable component while operating on a 66-MHz bus segment. This indicates that the function does not support the ***D2*** state.

It is the system software's responsibility to determine the clocking capabilities for all 66-MHz components on a given bus segment. If any of them cannot meet the 33-MHz clocking requirements while operating at 66 MHz, then the system software must ensure that the bus segment is never allowed to be placed into ***B2***.

66-MHz capable PCI components must first be capable of determining that the PCI bus that they reside on is running at 66 MHz if they wish to deviate from the 33-MHz clocking requirements while operating at 66 MHz. The most straightforward way of meeting this requirement is to add the **M66EN** signal (pin B49 of the PCI connector pinout) as an input to the 66-MHz capable component. When **M66EN** is a "1", it indicates that the bus segment clock frequency is 66 MHz. Components that are able to determine the bus clock frequency without knowledge of **M66EN** are free to do so by whatever other means available.

4.7 Control/Status of PCI Bus Power Management States

PCI bus power management states, as defined in this specification, adhere to the general policy that a bus's power state follows, or tracks, that of its Originating Device's power management state. So by writing to, or reading from, an originating bridge function's *PMCSR* **PowerState** field, the operating system can explicitly set, or determine its PCI bus's power management state.

Behavioral policy for power managed PCI functions on a given bus, as defined in Section 5.6, dictates that a PCI function must be at the same or greater power management state as that of the bus it physically resides on. So, for example, a PCI bridge whose secondary bus is in ***B1*** could correctly assume that no PCI functions on its secondary bus will attempt to initiate any bus traffic until the bridge's state is changed to ***D0*** which would result in the secondary bus's transition to ***B0***[6]. New PCI bridge designs could take advantage of this information regarding its surroundings to potentially achieve further power savings in their designs.

4.7.1 Control of Secondary Bus Power Source and Clock

This section defines the standard mechanism that system software uses to control the clock and power source of a PCI bus. This mechanism ties control of secondary bus power and clock to the Originating Device's power management state.

Table 12 defines the relationship between an originating PCI bridge function's power management state and that of its secondary bus. The third column defines actions that must occur as a direct consequence of the Originating Device's **PowerState** field having been programmed to the current power management state.

Table 12: PCI Bus Power and Clock Control

Originating Device's Bridge PM State	Secondary Bus PM States	Resultant Actions by Bridge (Either Direct or Indirect)
D0	***B0***	none
D1	***B1***	none
D2	***B2***	Clock stopped on secondary bus
$D3_{hot}$	***B2, B3***	Clock stopped and Vcc removed from secondary bus (***B3*** only). (Refer to the definition of **B2_B3#** in Table 8.)
$D3_{cold}$	***B3***	**none**

Note that the power management state of a PCI bus segment follows that of its originating bridge's power management state with one exception. The ***$D3_{hot}$*** state may cause its secondary bus's power management state to transition to either ***B2*** or ***B3***.

[6] Bus activity would not actually resume until the downstream functions were also programmed to states that permitted bus transactions to occur.

If a system designer choose not to implement a programmed ***B3*** bus power management state when the bridge function is transitioned to ***D3$_{hot}$***, the ***B2*** option allows the system to keep power applied to the bus segment while still achieving significant power savings (idle bus with clock stopped).

Two single bit read only fields defined in the function's *PMCSR_BSE* register support this functionality. The **B2_B3#** bit is used to determine the power management state of the secondary bus when its Originating Device's bridge function is transitioned to ***D3$_{hot}$***.

The **BPCC_En (Bus Power/Clock Control Enable)** bit serves as an enable/disable bit supported to allow other possibly system specific (ACPI, Hot-Plug, etc.) bus power and clock control schemes to bear the responsibility for controlling power and clock for a given PCI bus.

Chapter 5
PCI Function Power Management States

PCI defines a device as a physical load on the PCI bus. Each PCI device can host multiple functions each with its own PCI Configuration Space. Since each PCI function is an independent entity to the software, each function must implement its own power management interface. Each PCI function can be in one of four power management states. All PCI functions that adhere to this specification are required to support ***D0***, ***D3$_{hot}$***, and ***D3$_{cold}$***.

D1 and ***D2*** are optional power management states. These intermediate states are intended to afford the system designer more flexibility in balancing power savings, restore time, and low power feature availability tradeoffs for a given device class. The ***D1*** state could, for example, be supported as a slightly more power consuming state than ***D2***, however one that yields a quicker restore time than could be realized from ***D2***.

The ***D3*** power management state constitutes a special category of power management state in that a function could be transitioned into ***D3*** either by software or by physically removing power from its host PCI device. In that sense, the two ***D3*** variants have been designated as ***D3$_{hot}$*** and ***D3$_{cold}$*** where the subscript refers to the presence or absence of Vcc respectively. Functions in **D3$_{hot}$** can be transitioned to an uninitialized ***D0*** state via software by writing to the function's *PMCSR* register or by having its Bus Segment Reset (PCI **RST#**) asserted. Functions in the ***D3$_{cold}$*** state can only be transitioned to an uninitialized ***D0*** state by reapplying Vcc and asserting Bus Segment Reset (**RST#**) to the function's host PCI device.

PCI functions operating in either ***D0***, ***D1***, ***D2***, or ***D3$_{hot}$*** are required to be compliant with *PCI Local Bus Specification, Revision 2.1*.

5.1 PCI Function *D0* State

All PCI functions must support the ***D0*** state.

A PCI function must initially be put into ***D0*** before being used. Upon entering ***D0*** from power on reset, or transition from ***D3$_{hot}$***, the function will be in an uninitialized state. Once initialized by the system software, the function will be in the ***D0*** active state. All PCI functions must support ***D0***. A reset will force all PCI functions to the uninitialized ***D0*** state. Legacy PCI functions built prior to the release of this specification are assumed to be in ***D0*** whenever power is applied to them.

5.2 PCI Function *D1* State

Implementation of the ***D1*** power management state is optional.

D1 is used as a light sleep state. Some functions may be processing background tasks such as monitoring the network which actually requires most of the function to be active. In this state, the only PCI bus operation the PCI function is allowed to initiate is a PME. The function is only allowed to respond to PCI configuration accesses (i.e., memory and I/O spaces are disabled). Configuration Space must be accessible by system software while the function is in ***D1***.

If a function supports this state when operating on a 33-MHz PCI bus, it should also support it if operating on a 66-MHz PCI bus (if 66-MHz operation is supported).

5.3 PCI Function *D2* State

Implementation of the ***D2*** power management state is optional.

When a PCI function is not currently being used and probably will not be used for some time, it may be put into ***D2***. This state requires the function to provide significant power savings while still retaining the ability to fully recover to its previous condition. In this state, the only PCI bus operation the PCI function is allowed to initiate is a PME. The function is only allowed to respond to PCI configuration accesses (i.e., memory and I/O spaces are disabled). Configuration Space must be accessible by system software while the function is in ***D2***.

System software must restore the function to ***D0*** active before memory or I/O space can be accessed. Initiated actions such as bus mastering and functional interrupt request generation can only commence after the function has been restored to an active state.

There is a minimum recovery time requirement of 200 µs between when a function is programmed from ***D2*** to ***D0*** and when the function can be next accessed as a target (including PCI configuration accesses). If an access is attempted in violation of the specified minimum recovery time, undefined system behavior may result.

A function supporting this state when operating on a 33-MHz PCI bus is not required to support it if operating on a 66-MHz PCI bus (if 66-MHz operation is supported and the device is 66-MHz capable). Refer to Section 4.6.1 for more information.

5.4 PCI Function *D3* State

All PCI functions must support ***D3***.

In this state, function context need not be maintained. However, if PMEs are supported from ***D3***, then PME context must be retained at a minimum. When the function is brought back to ***D0*** (the only legal state transition from ***D3***), software will need to perform a full reinitialization of the function including its PCI Configuration Space.

There is a minimum recovery time requirement of 10 ms (enforced by system software) between when a function is programmed from ***D3*** to ***D0*** and when the function is accessed (including PCI configuration accesses). This allows time for the function to reset itself and bring itself to a power-on condition. It is important to note that regardless of whether the function is transitioned to ***D0*** from $\boldsymbol{D3_{hot}}$ or $\boldsymbol{D3_{cold}}$, the end result from a software perspective is that the function will be in the ***D0*** Uninitialized state.

5.4.1 Software Accessible D3 ($D3_{hot}$)

Functions in ***D3hot*** must respond to configuration space accesses as long as power and clock are supplied so that they can be returned to ***D0*** by software.

When programmed to ***D0***, the function performs the equivalent of a warm (soft) reset internally and returns to the ***D0*** Uninitialized state without PCI **RST#** being asserted. Other bus activity may be taking place during this time on the same PCI bus segment so the device that has returned to ***D0*** Uninitialized state must ensure that all of its PCI signal drivers remain disabled for the duration of the ***D3hot*** to ***D0*** Uninitialized state transition[7].

The only function context that must be retained in ***D3hot*** and through the soft reset transition to the ***D0*** Uninitialized state is the PME context.

5.4.2 Power Off ($D3_{cold}$)

If Vcc is removed from a PCI device, all of its PCI functions transition immediately to ***D3cold***. All PCI device functions support this state by default. When power is restored, PCI **RST#** must be asserted and functions will return to ***D0*** (***D0*** Uninitialized state) with a full PCI 2.1 compliant power-on reset sequence. Whenever the transition from ***D3*** to ***D0*** is initiated through assertion of PCI **RST#**, the power-on defaults will be restored to the function by hardware just as at initial power up. The function must then be fully initialized and reconfigured by software after making the transition to the ***D0*** uninitialized state.

Functions that support PMEs from ***D3cold*** must preserve their PME context through the ***D3cold*** to ***D0*** transition. The power required to do this must be provided by some auxiliary power source assuming that no power is made available to the PCI device from the normal Vcc power plane.

5.4.3 3.3Vaux / *$D3_{cold}$* Card Power Consumption Requirements

When the system is switched from its main supply outputs to the auxiliary power source, strict power budgeting with respect to which slots are allowed to consume full **3.3Vaux** power becomes necessary. A PCI function must draw no more than 20 mA through the **3.3Vaux** pin when in ***D3cold*** if its **PME_En** bit is cleared (i.e., 0b).

If a PCI function has been enabled for **PME#** generation (**PME_En** = 1b) prior to having entered into the ***D3cold*** state, the PCI add-in card (any single function or combination of multiple functions) may continue to draw up to 375 mA through the **3.3Vaux** pin while in ***D3cold***.

[7] PCI Bus signal drivers must behave the same as if the component had received a bus segment reset (**RST#**).

Auxiliary Power Consumption Reporting

The optional *Data Register* has been defined to enable the reporting of fine granular power utilization data for each of the supported power management states. In order to minimize implementation cost while also ensuring a robust architecture, a three bit field has been defined in the *PMC* register that is required for PCI functions that do not support the *Data Register* yet wish to draw from **3.3Vaux** while in ***D3$_{cold}$***.

PMC(8:6), ("**Aux_Current**"), provides a rudimentary power reporting vehicle for PCI functions to indicate their maximum required **3.3Vaux** current. Power budgeting software can then use this data to determine how many PCI functions can be configured for wakeup from ***D3$_{cold}$*** (refer to Table 6 for more information).

5.5 PCI Function Power State Transitions

All PCI function power management state changes are explicitly controlled by software except for hardware reset which brings all functions to the ***D0*** Uninitialized state. Figure 7 shows all supported state transitions. The unlabeled arcs represent a software initiated state transition (Set Power State operation).

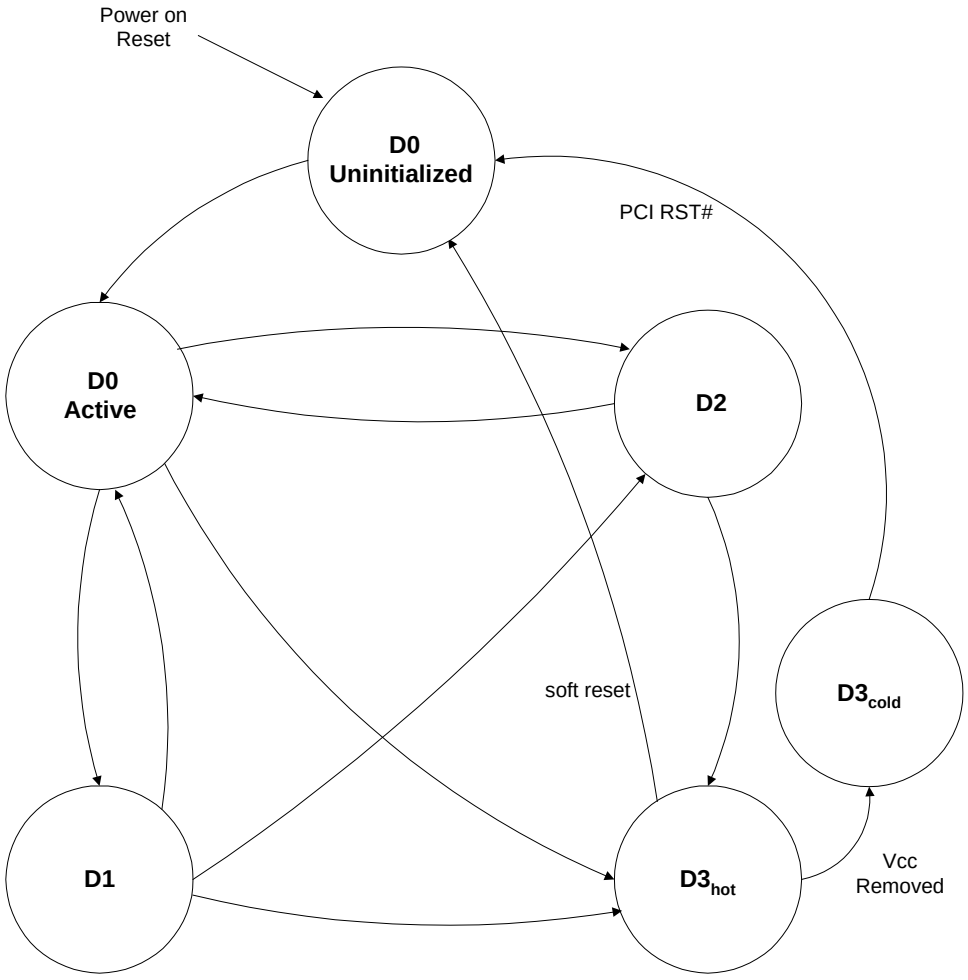


Figure 7: PCI Function Power Management State Transitions

5.6 PCI Function Power Management Policies

This section defines the behavior for PCI functions. Figure 8 illustrates the areas discussed in this section.

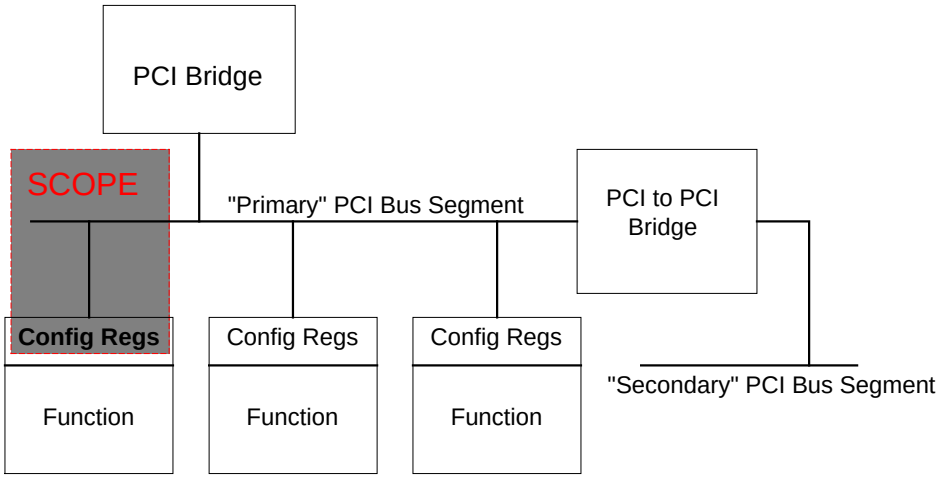


Figure 8: Non-Bridge PCI Function Power Management Diagram

Tables 13 to 17 define the behavior for a PCI function while operating in each combination of bus and functional power management states.

The columns of Tables 13 to 17 are defined as follows:

PCI Bus PM State — Current power management state of the PCI function's hosting PCI bus segment.

PCI Function PM State — Current PCI function power management state.

Context — Configuration register and functional state information that are required to be valid for the given power management state. The registers that must remain valid, and the features that must remain available for a given class of device, are typically dictated by the corresponding Device-Class Power Management specification.

Power — Power consumption.

Access Delay — The minimum required delay before attempting to access the PCI function to change its power state. If the bus is fully accessible (***B0***), then this delay is solely the result of the state transition delay, or recovery time, following the last write to the function's **PowerState** field. If the bus is not in a fully accessible state (***B1***, ***B2***, or ***B3***), then the delay is characterized by either the function's state transition recovery time, or the time it takes to restore the bus to a fully accessible state, whichever is greater.

Restore Time — The total time from when a PCI function transitions from its current power management state to the fully configured ***D0*** active state. (Measurement beginning from either a write to the function's *PMCSR* or a bus segment reset.)

Actions to Function — Valid PCI bus transactions that can be conducted with the function as the target of the transaction.

Actions from Function — Valid PCI bus transactions and/or operations that can be initiated by the function.

Table 13: D0 Power Management Policies

PCI Bus PM State	PCI Function PM State	Context	Power	Access Delay	Restore Time	Actions to Function	Actions from Function
B0	***Legacy PCI Function (D0)***	Full	Full	None	None	Any PCI transaction	Any PCI transaction or interrupt
B0	***D0 uninitialized***	PME Context *	<10 W	None	None	PCI configuration cycles	None
B0	***D0 (Active)***	Full	Full	None	None	Any PCI transaction	Any PCI transaction or interrupt, PME*
B1-B3	***D0 (Active)***	N/A**	N/A**	N/A**	N/A**	N/A**	N/A**

Notes:

* If PME is supported in this state.

** This combination of function and bus power management states is not allowed.

Table 14: D1 Power Management Policies

PCI Bus PM State	PCI Function PM State	Context	Power	Access Delay	Restore Time	Actions to Function	Actions from Function
B0	***D1***	Device-Class specific and PME Context*	≤ ***D0 uninitialized***	None	Device-Class specific	PCI configuration cycles and Device-Class specific	Device-Class specific and PME*
B1	***D1***	Device-Class specific and PME Context*	≤ ***D0 uninitialized***	Bus restoration time	Device-Class specific	None	PME only*
B2-B3	***D1***	N/A**	N/A**	N/A**	N/A**	N/A**	N/A**

Notes:

* If PME is supported in this state.

** This combination of function and bus power management states is not allowed.

Table 15: D2 Power Management Policies

PCI Bus PM State	PCI Function PM State	Context	Power	Access Delay	Restore Time	Actions to Function	Actions from Function
B0	***D2***	Device-Class specific and PME Context*	≤ Next lower supported PM state or ≤ ***D0 uninitial-ized***	200 ms (Note 1)	Device-Class specific	PCI configuration cycles	PME only*
B1	***D2***	Device-Class specific and PME Context*	≤ Next lower supported PM state or ≤ ***D0 uninitial-ized***	Greater of either the bus restoration time or 200 ms (Note 2)	Device-Class specific	none	PME only*
B2	***D2***	Device-Class specific and PME Context*	≤ Next lower supported PM state or ≤ ***D0 uninitial-ized***	Greater of either the bus restoration time or 200 ms (Note 2)	Device-Class specific	none	PME only*
B3	***D2***	N/A**	N/A**	N/A**	N/A**	N/A**	N/A**

Notes:

1. This condition is not typical. It specifies the case where the system software has programmed the function's **PowerState** field and then immediately decides to change its power state again. Typically, the state transition recovery time will have expired prior to a power state change request by software.

2. The more typical case where the bus must first be restored to ***B0*** before being able to access the function residing on the bus to request a change of its power state. State transition recovery time begins from the time of the last write to the function's **PowerState** field. In this case, the bus restoration time is dictated by state transition recovery times incurred in programming the bus's Originating Device to ***D0*** which then transitions its bus to ***B0***. Bus restoration time is typically the deciding factor in access delay for this case (refer to Section 5.6.1).

* If PME is supported in this state.

** This combination of function and bus power management states is not allowed.

Table 16: $D3_{hot}$ Power Management Policies

PCI Bus PM State	PCI Function PM State	Context	Power	Access Delay	Restore Time	Actions to Function	Actions from Function
B0	***$D3_{hot}$***	PME Context only*	≤ Next lower supported PM state or ≤ ***D0 uninitialized***	10 ms (Note 1)	Device-Class specific	PCI Config Cycles	PME only*
B1	***$D3_{hot}$***	PME Context only*	≤ Next lower supported PM state or ≤ ***D0 uninitialized***	Greater of either the bus restoration time or 10 ms (Note 2)	Device-Class specific	None	PME only*
B2	***$D3_{hot}$***	PME Context only*	≤ next lower supported PM state or ≤ ***D0 uninitialized***	Greater of either the bus restoration time or 10 ms (Note 2)	Device-Class specific	None	PME only*
B3	***$D3_{hot}$***	N/A**	N/A**	N/A**	N/A**	N/A**	N/A**

Notes:

1. This condition is not typical. It specifies the case where the system software has programmed the function's **PowerState** field and then immediately decides to change its power state again. Typically, the state transition recovery time will have expired prior to a power state change request by software.

2. The more typical case where the bus must first be restored to ***B0*** before being able to access the function residing on the bus to request a change of its power state. State transition recovery time begins from the time of the last write to the function's **PowerState** field. In this case, the bus restoration time is dictated by state transition recovery times incurred in programming the bus's Originating Device to ***D0*** which then transitions its bus to ***B0***. Bus restoration time is typically the deciding factor in access delay for this case (refer to Section 5.6.1).

* If PME is supported in this state.

** This combination of function and bus power management states is not allowed.

Table 17: *D3$_{cold}$* Power Management Policies

PCI Bus PM State	PCI Function PM State	Context	Power	Access Delay	Restore Time	Actions to Function	Actions from Function
B0-B2	***D3$_{cold}$***	N/A***	N/A***	N/A***	N/A***	N/A***	N/A***
B3	***D3$_{cold}$***	PME Context only*	No power from the bus	N/A	Full context restore or boot latency	Bus Segment Reset only	PME only*
B3	***Legacy PCI function (D3)***	None	No power	N/A	Full context restore or boot latency	Bus Segment Reset only	None

Notes:

1. This condition is not typical. It specifies the case where the system software has programmed the function's **PowerState** field and then immediately decides to change its power state again. Typically, the state transition recovery time will have expired prior to a power state change request by software.

2. The more typical case where the bus must first be restored to ***B0*** before being able to access the function residing on the bus to request a change of its power state. State transition recovery time begins from the time of the last write to the function's **PowerState** field. In this case, the bus restoration time is dictated by state transition recovery times incurred in programming the bus's Originating Device to ***D0*** which then transitions its bus to ***B0***. Bus restoration time is typically the deciding factor in access delay for this case (refer to Section 5.6.1).

* If PME is supported in this state.

** This combination of function and bus power management states is not allowed.

*** Implies device specific or slot specific power supplies which are outside the scope of this specification.

When in ***D2*** or ***D3$_{hot}$,*** a PCI function must not respond to PCI transactions targeting its I/O or memory spaces or assert a functional interrupt request.

A PCI function cannot tell the state of its PCI bus; therefore, it must always be ready to accept a PCI configuration access when in ***D1***, ***D2,*** or ***D3$_{hot}$***.

5.6.1 State Transition Recovery Time Requirements

Table 18 shows the minimum recovery times (delays) that must be guaranteed, by hardware in some cases and by system software in others, between the time that a function is programmed to change state and the time that the function is next accessed (including PCI Configuration Space). Note that for bridges, this delay also constitutes a minimum delay between when the bridge's state is upgraded to ***D0*** and when any function on the bus that it originates can be accessed.

Table 18: PCI Function State Transition Delays

Initial State	Next State	Minimum System Software Guaranteed Delays	Hardware Enforced Delays (by Bus's Originating Device)
D0	***D1***	0	N/A
D0 or ***D1***	***D2***	200 µs	≥ 16 PCI Clocks before the PCI Clock is stopped
D0, D1 or ***D2***	$D3_{hot}$	10 ms	≥ 16 PCI Clocks before the PCI Clock is stopped
D1	***D0***	0	N/A
D2	***D0***	200 µs	N/A
$D3_{hot}$	***D0***	10 ms	N/A

Chapter 6
PCI Bridges and Power Management

With power management under the direction of the operating system, each class of devices (PCI functions) must have a clearly defined criteria for feature availability as well as what functional context must be preserved when operating in each of the power management states. Some example Device-Class specifications have been proposed as part of the ACPI specification for various functions ranging from audio to network adapters. While defining Device-Class specific behavioral policies for most functions is well outside of this specification's scope, defining the required behavior for PCI bridge functions is within the scope of this specification. The definitions here apply to all three types of PCI bridges:

- Host bridge, PCI to expansion bus bridge, or other ACPI enumerated bridge
- PCI to PCI bridge
- A PCI to CardBus bridge

The mechanisms for controlling the state of these function vary somewhat depending on which type of Originating Device is present. The following sections describe how these mechanisms work for the three types of bridges.

This chapter details the power management policies for PCI bridge functions. The PCI bridge function can be characterized as an Originating Device with a secondary bus downstream of it. The relationship of the bridge function's power management state to that of its secondary bus has been mentioned in Section 4.7.1 and will be developed further in this chapter.

The shaded regions in Figure 9 illustrate what is discussed in this chapter.

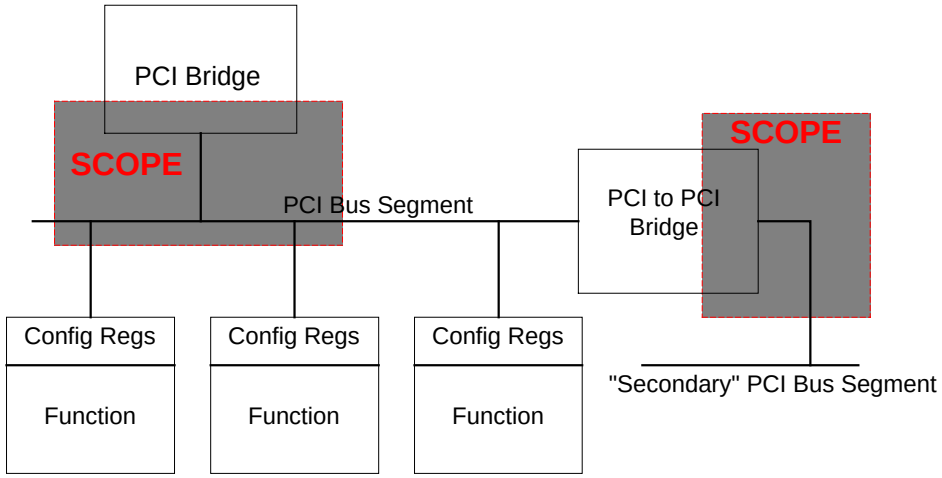


Figure 9: PCI Bridge Power Management Diagram

As can be seen from Figure 9, the PCI bridge behavior described in this chapter is common, from the perspective of the operating system, to both host bridges and PCI to PCI bridges.

The following table defines the relationship between a bridge function's power management state and that of its secondary bus. Also detailed are the resultant attributes of the secondary bus. The rightmost column of the table details a set of conditions that all *downstream* PCI functions must be capable of withstanding when residing on a bus in a given state without their application breaking in a way that cannot be gracefully recovered from.

It is the responsibility of the system software to ensure that only valid, workable combinations of bus and downstream PCI function power management states are used for a given PCI bus and all PCI functions residing on that bus.

The columns in Table 19 are defined are follows:

Bridge PM State	Current PCI function power management state of bridge function
Secondary Bus PM State	Current power management state of the Originating Device's (bridge function's) PCI bus segment
Secondary Bus Attributes	The characteristics of the secondary bus for the current bus power management state
Downstream Function Attributes	Necessary attributes of a PCI function residing on the secondary bus given the secondary bus's power management state

Table 19: PCI Bridge Power Management Policies

Bridge PM State	Secondary Bus PM State	Secondary Bus Attributes	Downstream Function Attributes
D0	***B0***	Any PCI transaction; clock and Vcc present	Any PCI 2.1 or 2.0 compliant function
D1	***B1***	No PCI transactions; clock and Vcc present	PCI function/application must continue to operate given that the PCI function cannot initiate any transactions, nor will it be targeted
D2	***B2***	No PCI transactions; no clock; Vcc present	PCI function/application must continue to operate given that the PCI function cannot initiate any transactions, nor will it be targeted. PCI function must not fail[8] without a PCI bus clock
$D3_{hot}$	***B2, B3***	No PCI transactions; no clock; Vcc may be present	All of the above plus potential absence of Vcc.
$D3_{cold}$	***B3***	No PCI transactions; no clock; no Vcc	Any PCI 2.1 or 2.0 compliant function

6.1 Host Bridge or Other Motherboard Enumerated Bridge

Bridges that cannot be supported on a standard PCI add-in card such as host bus bridges and docking bridges are not required to conform to the PCI Bus Power Management Interface Specification. However, they are encouraged to support functions states that are compatible with the PCI bus power management states defined within this specification. They are also encouraged to maintain their secondary bus at power management states which are compatible with the states defined above.

The mechanism for determining the capabilities of the bridge and controlling its power management state is motherboard specific. One example of a motherboard specific mechanism for providing operating system directed power management is the ACPI architecture. A bridge may implement the registers defined in this document and the motherboard specific mechanism could use those registers for part of the required functionality, but it is anticipated that in most cases, additional control of motherboard resources will be required to fully complete power management state transitions. Examples of this might be accesses to stop the clocks in the system and properly sequence the power supply.

[8] Failure does not include data overruns or underruns.

6.2 PCI to PCI Bridges

The power management policies for the secondary PCI bus of a PCI to PCI bridge are identical to those defined for any PCI bridge function.

The **BPCC_En** and **B2_B3#** bus power/clock control fields in the PCI to PCI bridge function's *PMCSR_BSE* register support the same functionality as for any other PCI bridges.

6.3 PCI to CardBus Bridge

While a PCI to CardBus bridge behaves much the same as a PCI to PCI bridge from a functional point-of-view, necessary definitional differences exist.

PCI to CardBus bridge designs should follow the PCMCIA's "PC-Card Standard". The PC-Card Standard consists of a series of specifications including:

- Electrical Specification
- Host System Specification
- PCI Bus Power Management Interface for CardBus Cards
- PCI Bus Power Management Interface Specification for PCI-to-CardBus Bridges

Chapter 7
Power Management Events

The Power Management Event (**PME#**) signal is an optional open drain, active low signal that is intended to be driven low by a PCI function to request a change in its current power management state and/or to indicate that a PME has occurred.

The **PME#** signal has been assigned to pin 19A of the standard PCI connector pinout (refer to Section 4.3.7. of the *PCI Local Bus Specification, Revision 2.1*).

The assertion and deassertion of **PME#** is asynchronous to the PCI clock.

In general, this signal is bused between all PCI connectors in a system although certain implementations may choose to pass separate buffered copies of the signal to the system logic. Functions must be enabled by software before asserting this signal. Once asserted, the device must continue to drive the signal low until software explicitly clears the **PME_Status** bit in the *PMSCR* register of the function. The system vendor must provide a pull-up on this signal if it allows the signal to be used. Systems vendors that do not use this signal are not required to bus it between connectors or provide pull-ups on those pins.

PME# is not intended to be used as an input by any PCI function, and the system is not required to route or buffer it in such a way that a function is guaranteed to detect that the signal has been asserted by another function.

Software will enable its use by setting the **PME_En** bit in the *PMCSR*. When a PCI function generates or detects an event which requires the system to change its power state (e.g., the phone rings), the function will assert **PME#.** It must continue to assert **PME#** until software either clears the **PME_En** bit or clears the **PME_Status** bit in the *PMCSR* even if the source of the PME is no longer valid (e.g., the phone stops ringing). Some devices which are powered by a battery or some external power source may use this signal even when powered off. Such devices must maintain the value of the **PME_Status** bit through reset.

PME# has additional electrical requirements over and above standard open drain signals that allow it to be shared between devices which are powered off and those which are powered on. The additional requirements include careful circuit design to ensure that a voltage applied to the **PME#** network will never cause damage to a component even if that particular component's Vcc inputs are not powered. Additionally, the device must ensure that it does not pull **PME#** low unless **PME#** is being intentionally asserted in all cases including when the function is in $\boldsymbol{D3_{cold}}$.

What this means is that any component implementing **PME#** must be designed such that:

- Unpowered **PME#** driver output circuits will not be damaged in the event that a voltage is applied to them from other powered wire-ORed sources of **PME#** .
- When power is removed from its **PME#** generation logic, the unpowered output does *not* present a low impedance path to ground or any other voltage.

These additional requirements ensure that the **PME#** signal network will continue to function properly when a mixture of powered and unpowered components have their **PME#** outputs wire-ORed together. It is extremely important to note that most commonly available open drain and tri-state buffer circuit designs used "as is" do NOT satisfy the additional circuit design requirements for **PME#.**

Other requirements on the motherboard/add-in card designer include:

- A common ground plane across the entire system.
- Split voltage power planes (Vaux vs. Vcc) are allowed.
- Vaux voltage supply must be present whenever AC power is applied to the system (if supported).
- Vcc at the PCI connector may be switched off by the system.

The card designer must be aware of the special requirements that constrain **PME#** and ensure that their card does not interfere with the proper operation of the **PME#** network. **PME#** input into the system may deassert as late as 100 ns after the **PME#** output from the function deasserts.

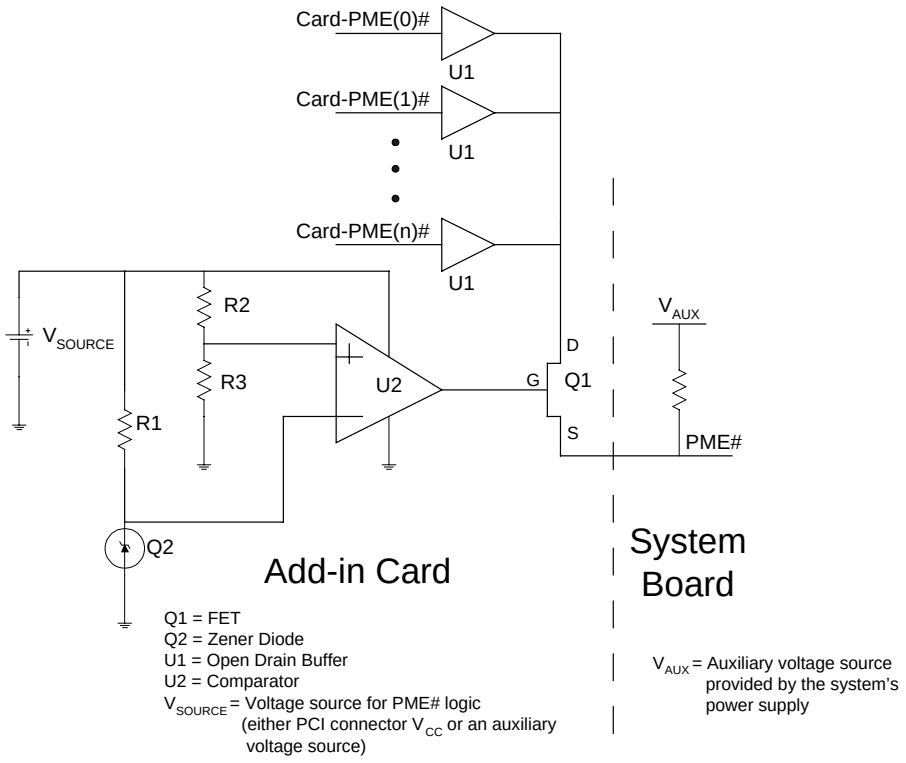
The value of the pull-up resistor for **PME#** on the system board should be derived taking into account the output capacitance of the open drain driver to ensure that **PME#** charges up to a logic high voltage level in no longer than 100 ns. (Refer to the *PCI Local Bus Specification* for information on pullup resistors.)

Implementation Note: Example PME# Circuit Design

The following diagram is an example of how the **PME#** generation logic could be implemented. In this example, multiple PCI functions, perhaps wire-ORed around a PCI to PCI bridge on the add-in card, have their ganged output connected to the single **PME#** pin on the PCI slot connector.

The circuit driving the gate of transistor Q1 is designed to isolate the card's **PME#** network from that of the system board whenever its power source (V_{SOURCE}) is absent.

If the card supplies power to its **PME#** logic with the PCI connector's V_{CC} (i.e., it does not support **PME#** from ***D3$_{cold}$***), then all of the **PME#** sources from the card will be isolated from the system board when the slot's V_{CC} is switched off. Cards that support **PME#** from ***D3$_{cold}$*** have an auxiliary power source to power the **PME#** logic which will maintain connection of these **PME#** sources to the system board network even when the bus power (V_{CC}) has been switched off.



This example assumes that *all* sources of **PME#** on the add-in card are powered by either V_{CC} or V_{AUX} (V_{SOURCE}). If **PME#** from ***D3$_{cold}$*** is supported by some, but not all of the card's functions that generate **PME#**, then the card designer must ensure that there is separate isolation control for each of the **PME#** generation power sources.

PCI component designers could choose to integrate the "power fail detect" isolation circuitry with their **PME#** output pin physically corresponding to the source of FET Q1. Alternatively, all isolation control logic could be implemented externally on the add-in card.

Note that this example is meant as a conceptual aid only and is not intended to prescribe an actual implementation.

7.1 Power Management Event (PME#) Signal Routing

The **PME#** signal is routed throughout the system in the same way that PCI bus functional interrupt requests are routed. All sources of **PME#** are typically connected together (wire-ORed) to present a single point of connection into the system.

This routing model imposes no backside or secondary bus impact on PCI to PCI bridge designs since "downstream" PCI functions' **PME#** signals are routed *around* the bridge and connected with the bridge's frontside or primary bus **PME#** signal network. Figure 10 illustrates the connection of **PME#** in a system with a PCI to PCI bridge.

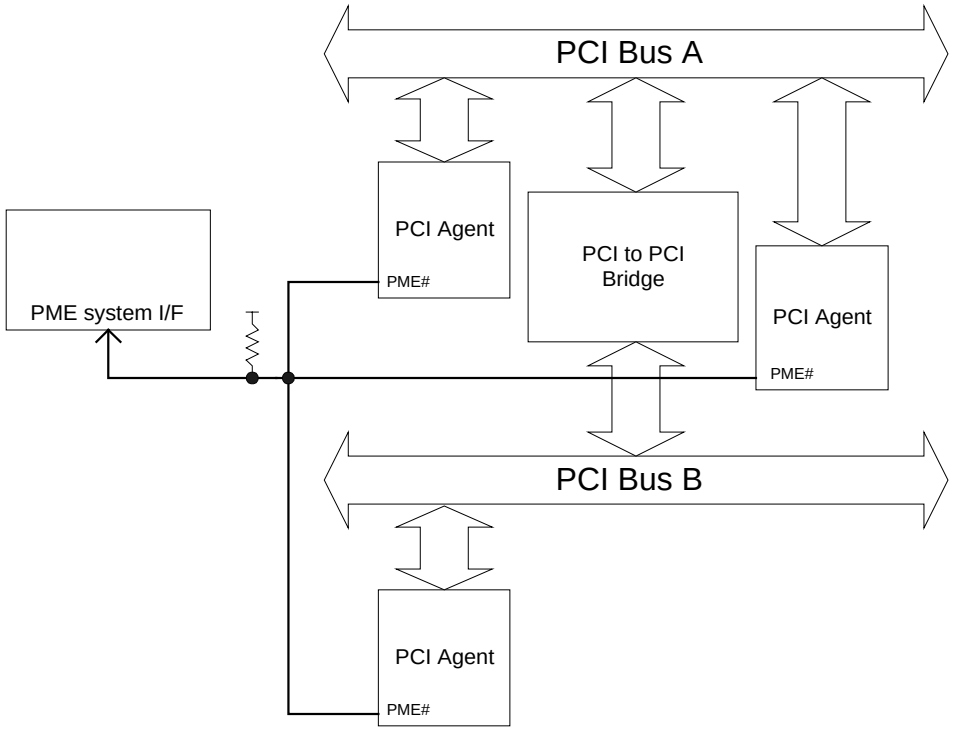


Figure 10: PME# System Routing

Note that while **PME#** is routed around most PCI to PCI bridges, PCI to CardBus bridges require that **PME#** must be routed through the bridge.

System software, having built itself a table of PCI functions that have been enabled for **PME#** generation, will walk this table at the start of the PME service routine to determine which PCI function, or functions, has requested a change in their power management state or for which a PME has occurred.

Large system designs, with correspondingly large PCI bus hierarchies, might consider a system specific way of presenting multiple parallel **PME#** subnet connections into the system as a means of potentially reducing the time it takes for system software to isolate which PCI function, or functions, has requested PME service. However, these implementations are beyond the scope of this specification.

Systems must route the **PME#** signal to the appropriate system logic to wake the system. For example, ACPI compliant systems may route this signal to the **SCI#** interrupt. Systems which support waking up from a "soft off" or a low power suspend where

significant portions of the system are powered off may route the signal to a power sequencing state machine.

7.2 Auxiliary Power

Power managed systems that support **PME#** generation while in the ***D3$_{cold}$*** state require an auxiliary power source given that the Vcc pins on the PCI expansion connector have been turned off. There are several ways to provide auxiliary power for any necessary *keep alive* circuitry including, but not limited to:

- On-board battery
- AC adapter, externally provided power source
- Auxiliary power supplied by the system

PCI systems, compliant with the *PCI Bus Power Management Interface Specification,* may support an optional auxiliary power source (**3.3Vaux**) connecting it to pin 14A of its PCI slot connectors.

7.2.1 3.3Vaux DC Characteristics:

PC system manufacturers who choose not to support **3.3Vaux** must leave pin 14A of each PCI slot connector floating (i.e., unconnected).

PC system manufacturers who choose to support **3.3Vaux** are required to physically route **3.3Vaux** to all PCI slots on their motherboard. Since most systems support a minimum of four PCI slots, it is essential to account for any auxiliary power required by the PCI slots and to budget the available auxiliary power appropriately.

Table 20 defines the DC operating environment that a **3.3Vaux** enabled system must deliver.

Table 20: DC Operating Environment for a 3.3Vaux-enabled System

Parameter	Min	Typ	Max	Units
3.3Vaux	3.0	3.3	3.6	Volts
$I_{MAX\_ENABLED}$	-	-	375 (Note 1)	mA
$I_{MAX\_DISABLED}$	-	-	20 (Note 2)	mA

Notes:

1. Upper limit when function is in ***D0***, ***D1***, ***D2***, ***D3$_{hot}$*** or is in ***D3$_{cold}$*** with its **PME_En** bit set.
2. Upper limit when function is in ***D3$_{cold}$*** and its **PME_En** bit is cleared to “0b”.

7.2.2 3.3Vaux Minimum Required Current Capacity

- ***B3* minimum requirement**

  At a minimum, systems supporting **3.3Vaux** must be capable of fully powering at least one **3.3Vaux** enabled PCI slot while the PCI bus is in ***B3***. In the case of a four PCI slot system for example, the minimum required **3.3Vaux** current capacity for that system would be 435 mA (one enabled slot and three disabled slots).

- ***B0*, *B1*, and *B2* minimum requirement**

  While in any bus powered state (i.e., ***B0***, ***B1***, or ***B2***), systems supporting **3.3Vaux** must be capable of fully powering all slots in the system. In the case of a four PCI slot system for example, the minimum required **3.3Vaux** current capacity for that system would be 1.5 A (four enabled PCI slots)

7.3 3.3Vaux System Design Requirements

System support for delivery of **3.3Vaux** to the PCI expansion slot connectors is optional. If, however, **3.3Vaux** is supported by the system, then all of the following requirements must be met.

7.3.1 Power Delivery Requirements

1. **3.3Vaux** must be connected to all motherboard PCI expansion slots.
2. The system must be capable of delivering up to 1.24 W (375 mA@3.3 VDC) to each ***enabled*** PCI expansion slot.
3. The system must be capable of delivering up to 66 mW (20 mA@3.3 VDC) to each ***disabled*** PCI expansion slot.
4. The system's auxiliary power source must be of sufficient capacity to support a minimum of one enabled PCI slot when the bus is in ***B3***.

PCI functions in ***D0***, ***D1***, ***D2***, or $\boldsymbol{D3_{hot}}$ are ***unconditionally enabled*** to consume up to the 1.24 W limit via **3.3Vaux**. This is consistent with systems that are designed with dual mode power sources. A dual mode power delivery subsystem supports two separately tuned (load-wise) power sources for the same voltage reference. In the case of 3.3V, for example, a dual mode power delivery subsystem will output either a high capacity 3.3V source for heavy "runtime" loads or a lower capacity 3.3V source for lightly loaded "sleeping" states.

3.3Vaux, when supported by the system, appears to each PCI connector pin 14A, as the logical "OR" of a high, and low capacity 3.3V source. In this way, under normally powered conditions (the PCI bus is in ***B0***, ***B1***, or ***B2***), all PCI add-in cards supporting $\boldsymbol{D3_{cold}}$ functionality connecting to pin 14A may draw up to 375 mA through the **3.3Vaux** pin. This voltage source "router" could be implemented discretely on the motherboard with power switches (FETs) and associated control logic, or it could be integrated into the power supply itself. Regardless of the approach to implementation, the voltage must never vary outside of the specified voltage regulation band (refer to Section 7.2) when switching between the main source and the lower capacity auxiliary source.

Software controls which slots are enabled for **PME#** generation from $\boldsymbol{D3_{cold}}$ and is responsible for remaining within the system's auxiliary power budget.

7.3.2 PCI Bus RST# Signaling Requirements

The section entitled "Reset" of the *PCI Local Bus Specification* requires the assertion of **RST#** whenever the PCI main power rails are out of spec, either when low and ramping on, or low as a result of a power failure event. In systems that do not support **3.3Vaux**, when a power failure event is occurring the assertion of **RST#** causes all PCI devices' output buffers to be placed in a high impedance state until such time that power is completely lost. At this point, the entire bus is floating including the **RST#** signal.

With the addition of "programmable power failures" and the **3.3Vaux** power source, the following new requirements for **RST#** are added to the existing requirements:

- The PCI Central Resource of systems that support **3.3Vaux** must ensure that the PCI **RST#** signal remains asserted whenever the PCI bus is in the *B3* state.

  This can be accomplished either by powering the PCI **RST#** output buffer with auxiliary power or by deploying a weak pulldown resistor on the **RST#** signal (on the motherboard) such that when power is lost at the **RST#** output buffer, the signal network will have a low impedance path to ground. With **RST#** asserted whenever the bus is in ***B3***, PCI functions that are in the $D3_{cold}$ state can depend on the low to high transition of **RST#** as a reliable indication that they must internally initialize all volatile portions of their function and transition themselves to the $D0_{uninitialized}$ state.

  Figure 11 illustrates the required behavior of **RST#** when the bus transitions into and out of ***B3.*** Refer to the "Reset" section of the *PCI Local Bus Specification* for existing **RST#** requirements.

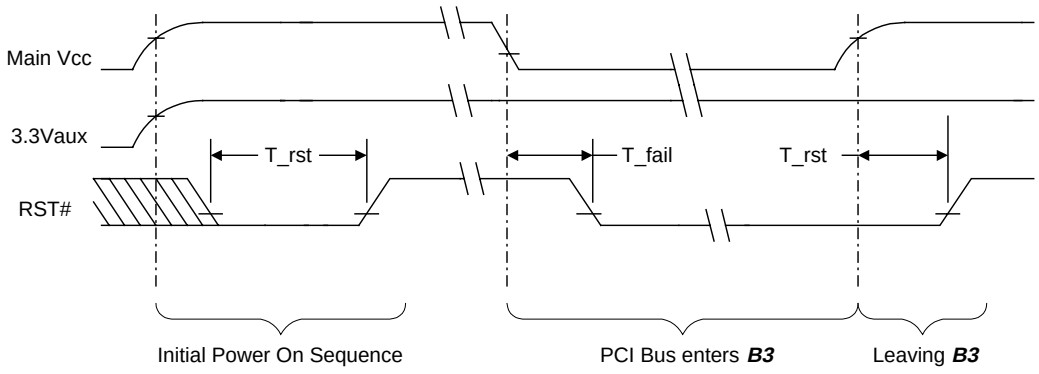


Figure 11: *B3* Reset Timing

- Functions in $D3_{cold}$ that are capable of generating **PME#** from $D3_{cold}$ must also be capable of accepting PCI bus cycles, including PCI configuration cycles, no later than 10 ms following the low to high transition of the PCI **RST#** signal.

  PCI functions that wake the system with power management events are designed specifically for quick resumption and must take no longer than 10 ms to self initialize to the point where they are ready to accept PCI cycles targeting them.

7.3.3 Voltage Sequencing

The **3.3Vaux** power source is independent of the PCI bus's 3.3V and 5V voltage rails, and there are no voltage timing sequencing requirements between them. This is consistent with the sequencing requirements in the *PCI Local Bus Specification.*

Once all supported voltages have settled to their specified levels, **3.3Vaux** must remain within its specified voltage regulation band unless the system has its input AC power source removed. This can occur in one of two ways:

- A mechanical switch is opened isolating the system from the input AC source.
- An AC power failure (power outage) has occurred.

7.4 3.3Vaux Add-in Card Design Requirements

7.4.1 3.3Vaux Power Consumption Requirements

Each **3.3Vaux** enabled PCI add-in card's load on **3.3Vaux** must not exceed 375 mA. This requirement applies to the PCI slot whether the load is represented by a single PCI function or by multiple functions. Likewise, when a function has entered the $D3_{cold}$ state and its **PME_En** bit has not been set (disabling it from generating **PME#**), it must then reduce its total slot **3.3Vaux** current consumption to less than or equal to 20 mA.

Implementation Note:

Reducing total slot current consumption to less than or equal to 20 mA can be accomplished by the add-in card in a number of ways ranging from internally disabling as much logic as possible to electrically isolating the **3.3Vaux** pin from its auxiliary power plane.

7.4.2 Physical Connection to the 3.3Vaux pin

Only PCI add-in cards implementing functions that support **PME#** from $D3_{cold}$ by design (i.e., functions that report PMC(15) = 1b) may physically connect to the **3.3Vaux** pin (14A). All other designs must not come into electrical contact with pin 14A and must only draw current from the main PCI bus voltage rails.

7.4.3 Isolation of 3.3Vaux from Main 3.3V

When a PCI add-in card is plugged into a system that supports **3.3Vaux**, the card must ensure that **3.3Vaux** is electrically isolated from the main PCI 3.3V rails at all times. This is done via split power planes with a separate power plane for **3.3Vaux** which never comes into electrical contact with the add-in card's main PCI 3.3V power plane.

In designs where logic paths cross between components powered by the **3.3Vaux** and 3.3V power domains, special care must be taken to ensure that when power is removed from the main 3.3V rail that no physical damage or logic malfunction occurs with respect to any of the subject devices be they powered or unpowered. The add-in card must electrically isolate these cross-domain powered circuit paths from each other as a direct result of being programmed to the $D3_{hot}$ state. While in $D3_{hot}$, all PCI power rails are still within their specified voltage regulation bands, and the $D3_{hot}$ state is always the initial step towards the eventual removal of main PCI bus power rails (i.e., ***B3*** / $D3_{cold}$).

7.4.4 3.3Vaux Presence Detection

PCI add-in cards implementing functions that can generate power management events from ***D3$_{cold}$*** must first determine the presence or absence of 3.3 volts on pin 14A before reporting their support for **PME#** from ***D3$_{cold}$***. A weak pulldown, attached to pin 14A, must be implemented on every PCI add-in card to create a logic low reference when plugged into a motherboard that does not support the delivery of **3.3Vaux**. Note that the current consumed by the pull down resistor must be included in the total slot **3.3Vaux** current budget. The function must then logically "AND" this reference with bit 15 of its *PMC* register when read back by the system. In this way, functions that normally do support **PME#** from ***D3$_{cold}$*** will report that they do not support it if plugged into a slot without **3.3Vaux**.

Compatibility with PCI Systems That Do Not Support 3.3Vaux

PCI add-in card manufacturers that target both the installed base of PCI systems as well as new PCI systems must ensure that, in a system that does not provide **3.3Vaux** to pin 14A, the card provides a means of powering its split auxiliary power plane using whatever other standard PCI voltage source it has available to it.

Figure 12 illustrates how an add-in card could route power to its auxiliary power plane based upon the presence or absence of **3.3Vaux** at pin 14A.

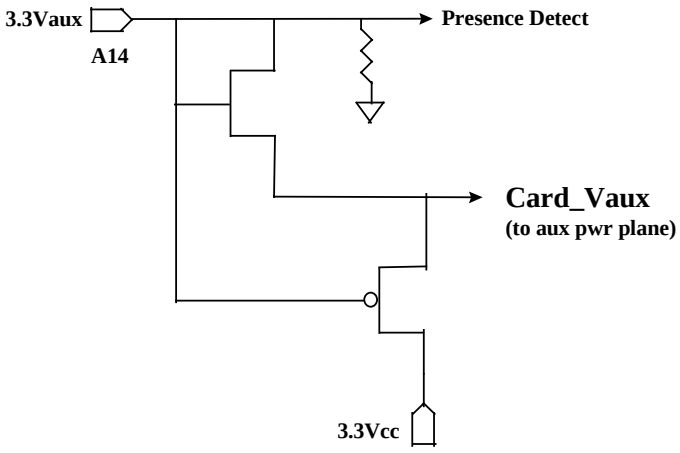


Figure 12: Add-in Card Auxiliary Power Routing

Chapter 8
Software Support for PCI Power Management

This specification defines the requirements for PCI functions to be managed by an operating system. The specification does not attempt to define any sort of Power Management Policy. That is left up to the individual operating system. However, it is necessary to address some of the basic assumptions that have been made regarding which aspects of power management are enforced by system software versus by hardware such that the functions might react appropriately. These assumptions fall into four basic categories:

- Identifying PCI function capabilities
- Placing PCI functions in a low power state
- Restoring PCI functions from a low power state
- Wake events

Within each of these areas, it has been assumed that the operating system software will handle certain power management functions in addition to its basic power management policy. It must be noted that in the context of this document, references to the operating system software include any device drivers and other operating system specific power management services.

8.1 Identifying PCI Function Capabilities

The operating system software is responsible for identifying all PCI function power capabilities by traversing the Capabilities structure in PCI Configuration Space and reading the Power Management Capabilities register (*PMC*). It is the operating system's responsibility to ensure that it does not attempt to place a function into a power management state which the function does not support. If, for any reason, the operating system software attempts to put a function into a power management state that the function does not support, the function should handle this gracefully[9] and remain in whatever state it was in before the request. The **PowerState** bits of the *PMCSR* will reflect the current state of the function, not the intended invalid state which was written.

[9] Finish the PCI transaction with normal completion and ignore the write data. This is *not* a hardware error condition.

8.2 Placing PCI Functions in a Low Power State

When attempting to place a PCI function in a low power state ***D1 - D3***, it is the operating system's responsibility to ensure that the function has no pending (host initiated) transactions, or in the case of a bridge device, that there are no PCI functions behind the bridge that require the bridge to be in the fully operational ***D0*** state. Furthermore, it is the operating system's responsibility to notify any and all device drivers that are conducting peer-to-peer transfers to the target function that the target function will no longer be accessible. In other words, it is the operating system's responsibility to ensure that no peer-to-peer activity occurs with the sleeping PCI function as the target. If the operating system and the PCI function both support Wake events, the operating system should enable the function's PME (**PME#**) line via the **PME_En** bit in the function's Power Management Control/Status Register (*PMCSR*).

8.2.1 Buses

When attempting to place a PCI bus segment into a lower power management state (***B1-B3***), the operating system must first ensure that all PCI functions on that bus have been placed in an appropriate low power state.

8.2.2 D3 State

Prior to placing a PCI function into ***D3***, the operating system must determine if the device driver for the function has the capability to restore the function from this state. Restoration includes reinitializing the function. PCI functions that require specific, non-standard, initialization may have the **DSI** bit (bit 5 in the *Power Management Capabilities* register) set. In the event that the device driver cannot restore the function from ***D3***, the operating system must make sure that the function is treated as a PCI legacy function. Note that the function's driver is responsible for fully restoring the function from ***D3***. If the device driver is not capable of fully reinitializing a function, the operating system should not put the function into ***D3***. When placing a function into ***D3***, the operating system software is required to disable I/O and memory space as well as bus mastering via the *PCI Command* register.

8.3 Restoring PCI Functions From a Low Power State

8.3.1 D0 "Uninitialized" and the DSI Bit

The **DSI** (Device Specific Initialization) bit (PMC bit 5) may be useful for some operating systems to properly support power management of add-in devices. Other operating systems, such as Microsoft Windows and Windows NT, use existing PCI-defined mechanisms (such as unique device IDs) to load device-specific drivers, and provide operating system-specific means to determine a driver's power management capabilities.

The types of PCI functions that may benefit from DSI support on some operating systems are those functions that require specific programming in order to appear as compatible device types. These are functions that power-up in one mode, and through configuration, can support additional modes.

Take, for example, a DSP audio controller that powers up into its native DSP mode. Through downloaded programming, this controller may be put into an "Audio Codec

'97" (AC '97) compatible mode. In order to support a variety of operating systems, the vendor provides an option ROM which programs the controller and places it into the AC '97 mode. This allows the controller to function in a compatible fashion and also allows it to function with the operating system's AC '97 compatible device driver if a native device driver is unavailable.

Other functions with similar "compatibility modes" will also require a device specific initialization.

PCI functions that do not support a compatibility mode and, therefore, will always have a device specific driver loaded, do not need to set the **DSI** bit.

The DSI may be consulted when the power management software wishes to put a function into the ***D3*** state. If a function has its **DSI** bit set, then the power management software could first ensure that it has the ability to restore this function to its initial state. The operating system has a device driver which specifically supports this function (i.e., *not* a compatible device driver), and thus the driver is capable of restoring the function.

All legacy PCI functions are assumed to require a device specific initialization sequence.

Note that the operating system may place any function in ***D3*** if it can guarantee that the system will execute a full system reset (POST) wherein all applicable Option ROM initialization code will be executed to restore functions to the operational state. This situation applies to the Soft Off system state.

8.3.2 D1 and D2 States

Restoring a function from ***D1*** or ***D2*** simply requires the operating system to update the **PowerState** field of the *PMCSR*. System software must ensure that sufficient time elapses between when the *PMCSR* is updated and when the first access targeting that PCI function may occur.

8.3.3 D3 State

Restoring a function from ***D3*** requires the operating system to reinitialize the function, beginning with, for the case of $\boldsymbol{D3_{cold}}$, restoring power to the device and initiating a PCI Bus Segment Reset. This is accomplished by either programming the hosting bus's Originating Device to ***D0*** or by other ACPI-type control methods.

Full context must be restored to the function before it is capable of resuming normal operation.

For example, reinitialization includes, but is not necessarily limited to, restoring the *Base Address* registers, re-enabling the I/O and memory spaces, re-enabling bus master capabilities, and unmasking any IRQs or PCI Interrupts as well as restoring the INT Line register. Furthermore, if the function has the DSI bit set, the operating system is required to execute whatever initialization code is necessary, either via the device driver's initialization code or by executing POST.

8.4 Wake Events

8.4.1 Wake Event Support

PCI power management supports Wake events generated by functions on the PCI bus. Assuming the operating system supports Wake events, it is the system hardware's responsibility to restore the host processor subsystem to a state which will permit the operating system to function (through ACPI or some other architecture). If the sub-system is already in a ***D0*** state, then the system hardware does not need to take any special action. The system is responsible for notifying the operating system that a PCI PME has occurred. It is expected that **PME#** will generate some form of System Configuration Interrupt (SCI), but whether this interrupt is handled by a device driver or an operating system service routine is left up to the individual operating system architecture.

Once the operating system has been notified that a PCI PME has occurred, it is the operating system's responsibility to restore power to the primary PCI bus and to restore it to the ***B0*** state, then to restore power to any unpowered slots/devices, and finally query the PCI functions that have been configured with **PME#** enabled to determine which function, or functions, had generated **PME#**. If the generating device is a bridge device, the operating system should follow this procedure for any subsequent PCI bridges. Should **PME#** become deasserted before the operating system identifies the device which generated it, or before the **PME#** is serviced, the operating system must recover gracefully. Furthermore, the operating system must be able to handle multiple **PME#s** generated by different functions simultaneously. Upon identifying the source or sources of the **PME#,** it is up to the operating system Power Management Policy to identify the correct course of action with regard to waking the functions and/or the rest of the system.

8.4.2 The D0 "Initialized" State From a Wake Event

Before the operating system returns a function to ***D0*** which will require a re-initialization of the function, it must ensure that the operating system not only has the information necessary to re-initialize the function, but also any information necessary to *restore the function* as well. This information is often client specific. As an example, assume a modem's client has set up a modem function in a specific state additional to default initialization (error correction, baud rate, modulation characteristics, etc.). The client/function then goes unused for an extended amount of time *which may cause* the power manager to place the modem in a ***D2*** or perhaps even a ***D3*** state.

When the client is called upon to interact with the modem (such as a ring-resume event), the operating system will have transitioned the modem function to the ***D0*** initialized state. However, restoration of the modem function to ***D0*** alone may not be sufficient for the function and client to perform the indicated task. It is manifest upon Device-Class specifications to include sufficient context save requirements for successful restoration of a function. The restoration must be transparent to the extent that the host application is unaware that a power state transition and the associated restoration occurred.

8.5 Get Capabilities

The Get Capabilities operation is performed by the operating system as it is enumerating functions in the system. Get Capabilities tells the operating system information about what power management features the function supports. This includes which power states the function supports, whether or not the function supports waking the machine, and from what power states it is capable of wakeup.

8.6 Set Power State

The Set Power State operation is used by the operating system to put a function into one of the four power states. The operating system will track the state of all functions in the system.

8.7 Get Power Status

The Get Power Status operation is used by the operating system to determine the present state of the power configuration (power states and features). This operation will read the *PMC* and *PMCSR* to obtain this information. Software assumes that reported status information reflects the current state of the function. Therefore, functions should update status information bits *only* after carrying out the intended task.

8.8 System BIOS Initialization

Since this specification was written to support operating system directed power management, there are minimal changes required by the system BIOS of the host system. While the system BIOS does not take an active role in power managing the PCI functions, the BIOS is still responsible for basic system initialization. PCI Power Management aware systems should have the following support in the system BIOS POST routines:

- During a true power-on situation, the BIOS should mask off the **PME#** signal through whatever system specific method is available. This is required because there is a small but finite chance that a **PME#** event might occur during the power up processes, before a **PME#** handler has been loaded.
- During a warm-boot (Control-Alt-Del) sequence, the BIOS should likewise mask off **PME#** events until the **PME#** event handler loads. This is required because there is a small but finite chance that a function enabled for **PME#** wakeup might have cause to generate a **PME#** during the power up processes, before a **PME#** handler has been loaded.
- During a warm-boot (Control-Alt-Del) sequence, the BIOS must restore all PCI Power Managed functions to ***D0***. Functions placed in the ***D1-D3*** states will remain in that state until programmed otherwise or until a PCI reset **(RST#)** occurs. PCI **RST#** assertion is the preferred method for causing all PCI functions to transition to ***D0***. If this cannot be accomplished, the BIOS must individually program each of the PCI function's *PMCSR* **PowerState** fields to ***D0***.

Chapter 9
Other Considerations

In addition to supporting the minimum set of required mechanisms defined in this specification, designers of PCI devices and systems are encouraged to add additional power management functionality, taking advantage of the optional headroom provided by this specification.

Designers are encouraged to design for low power consumption in all operating modes. The *PCI Mobile Design Guide* makes several suggestions on designing for low power which are applicable to all devices. Even simple things such as minimizing current drain through pull-up resistors can add up to real power savings.

Additional power saving techniques while in ***D0*** are also encouraged as long as they are transparent to the operating system.